9-12-2021

Optimal Steering of Nonredundant CMG Configurations for Spacecraft Attitude Control

Victor Hakim

Santa Clara University

Department of Mechanical Engineering

Date: September 12, 2021

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Victor Hakim

ENTITLED

Optimal Steering of Nonredundant CMG Configurations for Spacecraft Attitude Control

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

Mohammad Ayoubi

Thesis Advisor
Dr. Mohammad A. Ayoubi

Michael Taylor

Thesis Reader
Dr. Michael Taylor

Chairman of Department
Dr. Hohyun Lee

Optimal Steering of Nonredundant CMG Configurations for Spacecraft Attitude Control

By

Victor Hakim

Dissertation

Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Mechanical Engineering
in the School of Engineering at
Santa Clara University, 2021

Santa Clara, California

Acknowledgments

I wish to first and foremost thank my research advisor, Professor Mohammad Ayoubi of Santa Clara University. I thank him for his guidance in research, for his being my teacher in so many important courses, for his patience in my studies, and for his well-appreciated feedback on all of my work. I've learned so much from him in not only spacecraft control but also sheer research skills. I am exceedingly grateful for his teaching me and providing me this opportunity to dive into spacecraft attitude control.

I would also like to thank my aunt, Lingling Song, for her financial contribution to my tuition cost.

Optimal Steering of Nonredundant CMG Configurations for Spacecraft Attitude Control

Victor Hakim

Department of Mechanical Engineering
Santa Clara University
Santa Clara, California
2021

ABSTRACT

This thesis presents the problem of optimal steering with three or two single-gimbal control moment gyros (SGCMGs). Most SGCMG configurations use more than three rotors so that the resulting Jacobian always has a nontrivial nullspace. We consider the specific case of a four-CMG pyramid configuration where only three or two CMGs are operational. We use the Gauss pseudospectral method to find the optimal gimbal angle trajectories of the CMGs and the states of an agile spacecraft subject to mission-specific constraints. We do so while either minimizing time or maximally avoiding the angular-momentum singularities. The proposed steering method is implemented in scenarios where the gimbal angles initially start in both nonsingular and singular positions. A closed-loop quaternion feedback steering law is also developed to reduce total CMG angular momentum, which ensures solution existence in the case of two CMGs. Results are compared with the standard four-SGCMG maneuvering via the generalized singularity robust steering law in the literature.

Table of Contents

List of Figures

List of Tables

Nomenclature

Number of CMGs in configuration	n
Skew angle of pyramid configuration	β
Spacecraft tensor of inertia	J
Spacecraft angular velocity	$\boldsymbol{\omega}$
Spacecraft quaternion	$\mathbf{q}$
CMG configuration total momentum	$\boldsymbol{h}$
Individual CMG momentum	$\boldsymbol{h}_i$
Gimbal angles	$\boldsymbol{\delta}$
Individual gimbal angle	δ_i
Each CMG axial momentum magnitude	η
Jacobian	A
Cost function	C
"is defined as"	$\doteq$

Chapter 1

Background

For the past half-century, control moment gyroscopes (CMGs) have been studied extensively and used for the attitude control of various spacecraft and space missions. CMGs are often chosen as the primary means of attitude control in agile spacecraft because of their high torque output. A CMG contains a spinning rotor which is kept rotating at a constant angular speed but whose angular momentum vector changes direction relative to the spacecraft by gimballing the spinning rotor. Reaction wheels differ from CMGs by instead keeping their angular momentum vector directions constant and varying their angular speeds to apply torque. Reaction wheels have the advantage of mathematical simplicity, though their maximum torque is typically less than 2 N·m [1]. CMGs, on the other hand, can have a range of 100-5000 N·m of maximum torque [1]. The CMG has a high torque capability because it is a torque amplification device—a small gimbal torque input produces a large control torque output on the spacecraft [1, 2]. Thus CMGs are usually preferred in missions where spacecraft agility is desired.

CMGs can be single-gimbal (SGCMG) or double-gimbal (DGCMG). In DGCMGs, the angular momentum vector can point along any direction in the sphere, while SGCMGs allow only planar pivoting of the angular momentum vector about a gimbal axis, orthogonal to the angular momentum vector. DGCMGs offer the advantage of simpler singularity avoidance, but they have the disadvantage of more difficult hardware assembly. In recent decades, several types of SGCMG steering logic have been proposed which can avoid singularities [3], and so faithful SGCMG implementations are becom-

ing more common. Most SGCMG configurations would use $n \geq 4$ rotors in order to be redundant; that is, the resulting Jacobian would always have a nontrivial nullspace. With a nontrivial nullspace, the gimbal angles can be varied without producing an output torque on the spacecraft body, and this so-called "null motion" helps to steer the gimbal angles out of or away from singularities without excessive deviation from the commanded torque. The gimbal angles are in a singularity if the possible output torque, which should ideally span three dimensions, is reduced in dimensionality to two (or one) dimensions. Avoiding singularities is the main mathematical difficulty to be addressed in any SGCMG steering method. Since SGCMGs are the predominant focus of study, we will hereafter take "CMG" to mean "SGCMG."

Most of the literature [1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14] focuses on analyzing the four-CMG pyramid configuration, depicted in Figure 1.1. Clearly, an $n = 4$ configuration is advantageous because it is the most mass-efficient while still being redundant, as long as the steering algorithm can effectively deal with singular states. However, if one CMG rotor fails, then the configuration becomes nonredundant, and singularity avoidance is much more difficult. CMGs have failed in past missions, for example on the ISS [15] and more recently on WorldView-4 [16]. It is still possible to provide three-axis attitude control on the spacecraft as long as there are at least three functioning CMG rotors with independent gimbal axes. Not much literature is devoted to gimbal steering in the case of three-CMG configurations. Sands et al. [14] propose a method by constraining the momentum envelope, but the method requires the skew angle β to be varied, which isn't usually possible in SGCMG designs. Lee and Bang [6] provide "preferred" gimbal angles for three-CMG configurations, a useful construct in developing a steering law. This thesis deals with the case of a four-CMG pyramid configuration (Figure 1.1) where one or two CMGs have failed and only three or two CMGs are operational. (We'll call these the **3/4 pyramid configuration** and the **2/4 pyramid configuration**, respectively.)

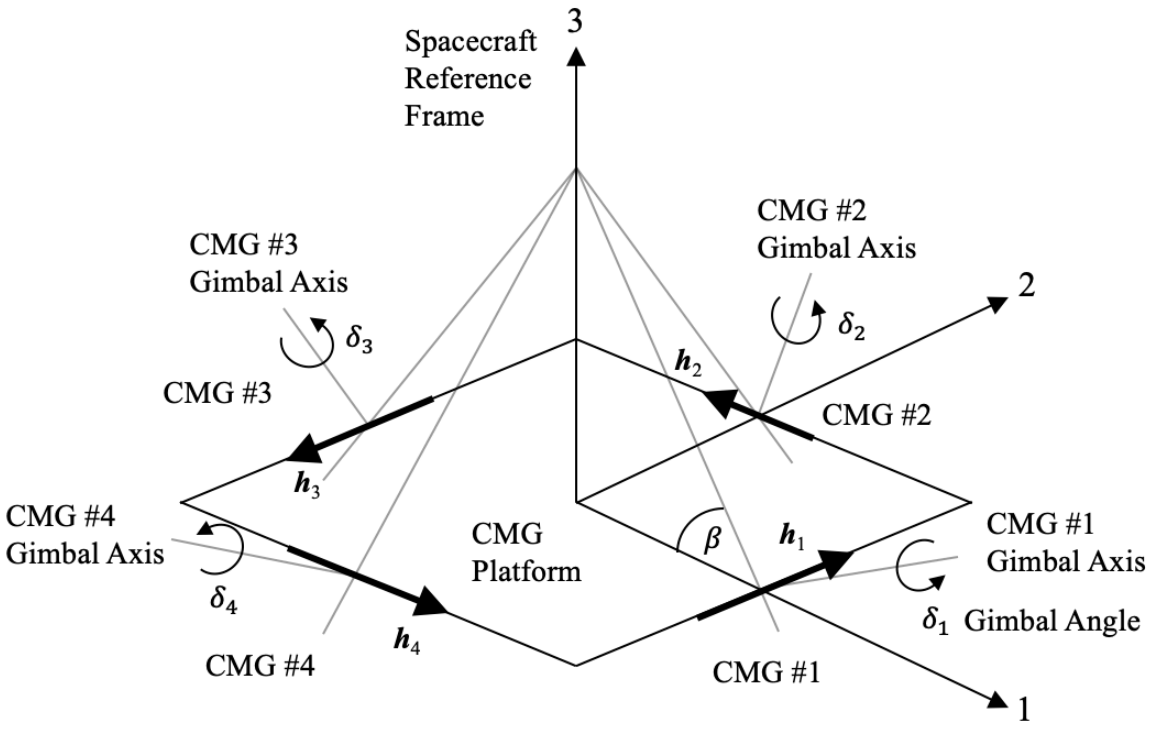


Fig. 1.1: Four-CMG Pyramid Configuration [4]

In Chapter 2, we compute optimal gimbal angle trajectories for spacecraft equipped with a nonredundant 3/4 CMG pyramid configuration. We compute the gimbal angle trajectories using the Gauss pseudospectral method [17, 18]. Pseudospectral methods are direct methods which have been useful for computational optimal control. The states' and controls' evolution over time are approximated with Lagrange interpolating polynomials collocated at certain node points. Computation of the optimal state and control values at the node points becomes a nonlinear programming problem (NLP). In the Gauss pseudospectral method, the node points are at the Legendre-Gauss (LG) points.

Optimal gimbal angle trajectory planning has been used before on CMGs [8, 9, 10, 11, 13]. Paradiso [9] used a directed search method. Fleming and Ross [8], Sun et al. [10], and Zhang et al. [11] have used pseudospectral methods for optimization of some

cost function. Fleming and Ross [8], however, provide no constraint on the singularity measure of the Jacobian, thus the method cannot guarantee that singularity will be maximally avoided during the maneuver. All of [8, 9, 10, 11] focus on redundant ($n \geq 4$) CMG configurations. Lee et al. [13] compute optimal gimbal angle trajectories for nonredundant CMG configurations and maximize the integrated distance from singularity, though one drawback of their method is that the final time is not free.

In Chapter 2, we use the Gauss pseudospectral method to compute gimbal angle trajectories which perform a desired spacecraft slew in the failure case of a 3/4 CMG configuration. In Chapter 3, we discuss how the 2/4 CMG configuration adds further complications with solution existence. That is, due to the CMG momentum envelope in the 2/4 CMG configuration being severely deflated and non-spherical, certain spacecraft rotations would be impossible if the CMG momentum is saturated. Chapter 3 demonstrates this impossibility and provides a steering algorithm to reduce the CMG angular momentum using an external torque. In Chapter 4, we then assume a 2/4 CMG configuration which is unsaturated and use the Gauss pseudospectral method to compute gimbal angle trajectories for a desired spacecraft slew. Finally, concluding remarks are provided in Chapter 5.

In this thesis, quaternions use q_4 as the scalar component. In addition, some of the code used to implement the Gauss pseudospectral method was modified from similar-method optimal control MATLAB code provided by Daniel Herber [19]. MATLAB codes are provided in Appendices B - J.

1.1 Equations of Motion

Consider a spacecraft rotating within an inertial reference frame. Fix a non-rotating reference frame onto the spacecraft so that the origin always coincides with the space-

craft's center of mass; we'll call this the "space frame." (Note that this reference frame is generally noninertial because the spacecraft center of mass accelerates.) Attach a co-rotating reference frame to the body of the spacecraft with the origin at the spacecraft's center of mass; we'll call this the "body frame." The body frame rotates exactly with the spacecraft's rotation. Since we are only concerned with the spacecraft's orientation and not its position within the inertial frame, we only need to consider the space frame and body frame.

Let $\boldsymbol{\omega} = (\omega_1, \omega_2, \omega_3)^T$ be the angular velocity vector of the spacecraft, given in body frame coordinates. Let $\mathbf{q} = (q_1, q_2, q_3, q_4)^T$ be the quaternion that describes the orientation of the spacecraft (or body frame) relative to the space frame. Here, q_4 is the scalar component and $\mathbf{q}_v \doteq (q_1, q_2, q_3)^T$ is the vector component. The kinematic equations of motion are given by [20]

$$\dot{\mathbf{q}}_v = -\tfrac{1}{2}\boldsymbol{\omega} \times \mathbf{q}_v + \tfrac{1}{2} q_4 \boldsymbol{\omega} \tag{1.1a}$$

$$\dot{q}_4 = -\tfrac{1}{2}\boldsymbol{\omega} \cdot \mathbf{q}_v. \tag{1.1b}$$

Given the external torque $\mathbf{T}_{\text{ext}}$ in the body frame, the evolution of $\boldsymbol{\omega}$ is given by

$$J\dot{\boldsymbol{\omega}} + \dot{\boldsymbol{h}} + \boldsymbol{\omega} \times \big(J\boldsymbol{\omega} + \boldsymbol{h}\big) = \mathbf{T}_{\text{ext}}. \tag{1.2}$$

Here, J is the spacecraft's tensor of inertia, with coordinates given in the body frame, which includes the CMG rotors. $\boldsymbol{h}$ is the total angular momentum of all CMG rotors *from the perspective of a bystander fixed on the spacecraft and rotating with the spacecraft.* Note that this is *not* the same as the total angular momentum of the CMG rotors with coordinates taken in the body frame. (This is because $\boldsymbol{h}$ does not include the rotation of each CMG rotor's center of mass about the spacecraft center of mass.) See Appendix A for a clarification on the meaning of the spacecraft tensor J and a

derivation of equation 1.2.

In Chapter 2, we will focus on the four-CMG pyramid configuration, shown in Figure 1.1, except with one CMG rotor in failure. Without loss of generality, let CMG #4 be in failure. Let $\boldsymbol{\delta} = (\delta_1, \delta_2, \delta_3)^T$ be a vector which lists the CMG gimbal angles. Assume each CMG's gimbal rate $\dot{\delta}_i$ is negligible compared to its rotor angular velocity. (This is equivalent to ignoring $\ddot{\delta}_i$ in $\dot{\boldsymbol{h}}$ within other literature.) Also assume that each CMG rotor's center of mass is fixed within the spacecraft at all times. Also assume each CMG rotor's axial direction is a principal axis of the rotor, and let each CMG rotor have the same constant axial angular momentum magnitude η. In this case, $\boldsymbol{h}$ is given by the summation over the three functioning CMGs

$$\begin{aligned} \boldsymbol{h} &= \boldsymbol{h}_1(\delta_1) + \boldsymbol{h}_2(\delta_2) + \boldsymbol{h}_3(\delta_3) \\ &= \eta \left(\begin{bmatrix} -c\beta\, s_1 \\ c_1 \\ s\beta\, s_1 \end{bmatrix} + \begin{bmatrix} -c_2 \\ -c\beta\, s_2 \\ s\beta\, s_2 \end{bmatrix} + \begin{bmatrix} c\beta\, s_3 \\ -c_3 \\ s\beta\, s_3 \end{bmatrix} \right), \end{aligned} \tag{1.3}$$

where $c\beta \doteq \cos\beta$, $c_i \doteq \cos(\delta_i)$, $s\beta \doteq \sin\beta$, and $s_i \doteq \sin(\delta_i)$. Then $\dot{\boldsymbol{h}}$ is

$$\dot{\boldsymbol{h}} = A\dot{\boldsymbol{\delta}}. \tag{1.4}$$

Here, A is called the *Jacobian*, given by

$$A = \eta \begin{bmatrix} -c\beta\, c_1 & s_2 & c\beta\, c_3 \\ -s_1 & -c\beta\, c_2 & s_3 \\ s\beta\, c_1 & s\beta\, c_2 & s\beta\, c_3 \end{bmatrix}. \tag{1.5}$$

The gimbal angles are in a singular position exactly when the Jacobian has a rank less than 3. In our case, since A is square, the Jacobian is singular if and only if $\det(A) = 0$.

In the absence of external torques, equation (1.2) becomes

$$J\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \left(J\boldsymbol{\omega} + \boldsymbol{h}\right) = -A\dot{\boldsymbol{\delta}}. \tag{1.6}$$

The right-hand side can be thought of as the "effective torque" applied onto the spacecraft due to the CMG gimbal angles' maneuvering. It is clear that when the gimbal angles are singular (equivalently the Jacobian A is singular, equivalently A has rank less than 3), the CMG assembly is only able to apply torque over at most a two-dimensional space, and this is undesirable. Thus there is the need for the gimbal angles to avoid singularity as much as possible.

Ultimately, a steering law is chosen to control the gimbal angles in such a way as to slew or point the spacecraft attitude as desired. Figure 1.2 summarizes how a steering law's control of gimbal angles affects the spacecraft's attitude.

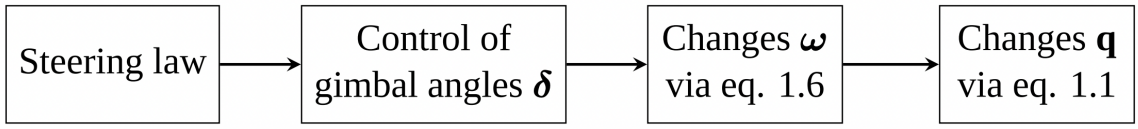


Fig. 1.2: Summary of how a steering law controls spacecraft attitude

In the case of the 2/4 CMG configuration, where there are only two functioning CMGs, the Jacobian (equation 1.5) loses a column, corresponding to the gimbal which is unused. Chapters 3 and 4 consider this extreme failure case.

1.2 The Gauss Pseudospectral Method

There are two main general categories of methods to numerically solve optimal control problems: direct and indirect. Indirect methods will minimize cost by finding approximate solutions that satisfy the first-order necessary conditions for optimality

[21]. Direct methods transcribe the equations of motion, equality constraints, and inequality constraints into finite pieces which can be directly optimized by a nonlinear programming (NLP) method [21]. For a comprehensive survey on the various methods for numerical optimal control, see Betts [22]. Pseudospectral methods are direct methods which collocate the states and controls at a finite number of nodes within the given time interval. Many pseudospectral methods exist; they are distinguished by the choice of nodes on which the states and controls are collocated. See Hart et al. [21] for an excellent comparison of different pseudospectral methods. This thesis chooses to use the Gauss pseudospectral method, one of the more recently developed methods [17, 18]. In practice, any of the pseudospectral methods could likely be adapted to solve the nonredundant CMG problem.

In this section, we describe the Gauss pseudospectral method [17, 18]. First, the continuous Bolza problem with time $t \in [t_0, t_f]$ is transformed into the normalized time $\tau \in [-1, 1]$ with the affine transformation

$$t = \frac{t_f - t_0}{2}\tau + \frac{t_f + t_0}{2}. \tag{1.7}$$

We then wish to minimize the cost functional, in general given by

$$C = \Phi[\mathbf{x}(-1), \mathbf{x}(1)] + \frac{t_f - t_0}{2} \int_{-1}^{1} g[\mathbf{x}(\tau), \mathbf{u}(\tau)]\, d\tau. \tag{1.8}$$

We must determine the state $\mathbf{x}(\tau) \in \mathbb{R}^r$ and the control $\mathbf{u}(\tau) \in \mathbb{R}^m$ which minimizes C. The initial time t_0 and final time t_f may be fixed or free. The state and control are

subject to the general constraints

$$\begin{aligned}
\frac{d\mathbf{x}}{d\tau} =& \frac{t_f - t_0}{2}\,\mathbf{f}[\mathbf{x}(\tau), \mathbf{u}(\tau)] \in \mathbb{R}^r && \text{(diff. equation)} \\
\boldsymbol{\phi}\,[\mathbf{x}(-1), \mathbf{x}(1)] &= \mathbf{0} \in \mathbb{R}^q && \text{(boundary conditions)} \\
\boldsymbol{\Gamma}\,[\mathbf{x}(\tau), \mathbf{u}(\tau)] &\leq \mathbf{0} \in \mathbb{R}^c. && \text{(path constraints)}
\end{aligned}$$

If we wish to have N internal node points, we use the Legendre-Gauss (LG) points, i.e. the roots of the Nth degree Legendre polynomial, ordered such that

$$-1 < \tau_1 < \tau_2 < \cdots < \tau_N < 1.$$

Let $\tau_0 = -1$ and $L_i(\tau)$ $(i = 0, 1, 2, ..., N)$ be the Lagrange interpolating polynomials

$$L_i(\tau) = \prod_{j=0, j\neq i}^{N} \frac{\tau - \tau_j}{\tau_i - \tau_j} \ (i = 0, 1, 2, ..., N). \tag{1.9}$$

Note that $L_i(\tau_j)$ equals 1 if $i = j$ and 0 otherwise. We approximate the state as

$$\mathbf{x}(\tau) \approx \mathbf{X}(\tau) \doteq \sum_{i=0}^{N} \mathbf{X}(\tau_i) L_i(\tau) = \sum_{i=0}^{N} \mathbf{X}_i L_i(\tau). \tag{1.10}$$

Now let $L_i^*(\tau)$ $(i = 1, 2, ..., N)$ be defined as

$$L_i^*(\tau) = \prod_{j=1, j\neq i}^{N} \frac{\tau - \tau_j}{\tau_i - \tau_j} \ (i = 1, 2, ..., N). \tag{1.11}$$

(Note the index starts at 1 instead of 0.) We approximate the control as

$$\mathbf{u}(\tau) \approx \mathbf{U}(\tau) \doteq \sum_{i=1}^{N} \mathbf{U}(\tau_i) L_i^*(\tau) = \sum_{i=1}^{N} \mathbf{U}_i L_i^*(\tau). \tag{1.12}$$

Differentiate $\mathbf{X}(\tau)$ to obtain an approximation for $\dot{\mathbf{x}}(\tau)$

$$\dot{\mathbf{x}}(\tau) \approx \dot{\mathbf{X}}(\tau) = \sum_{i=0}^{N} \mathbf{X}_i \dot{L}_i(\tau). \tag{1.13}$$

Now, we define the so-called differential approximation matrix $D \in \mathbb{R}^{N \times N+1}$ with elements

$$D_{ki} = \dot{L}_i(\tau_k), \tag{1.14}$$

where $k = 1, 2, ..., N$ and $i = 0, 1, 2, ..., N$. The differential equation is then approximated as

$$\sum_{i=0}^{N} D_{ki} \mathbf{X}_i = \frac{t_f - t_0}{2} \mathbf{f}(\mathbf{X}_k, \mathbf{U}_k, \tau_k) \ (k = 1, 2, ..., N). \tag{1.15}$$

We approximate $\mathbf{x}(\tau = 1)$ with $\mathbf{X}_f$, defined by using a Gauss quadrature

$$\mathbf{X}_f \doteq \mathbf{X}_0 + \frac{t_f - t_0}{2} \sum_{k=1}^{N} w_k \, \mathbf{f}(\mathbf{X}_k, \mathbf{U}_k, \tau_k), \tag{1.16}$$

where w_k are the Gauss weights, given by

$$w_k = \frac{2}{\left(1 - \tau_k^2\right) \left[\dot{P}_N(\tau_k)\right]^2} \quad (k = 1, ..., N), \tag{1.17}$$

where $P_N(\tau)$ is the Nth degree Legendre polynomial. The cost C is approximated also using a Gauss quadrature

$$C = \Phi(\mathbf{X}_0, \mathbf{X}_f) + \frac{t_f - t_0}{2} \sum_{k=1}^{N} w_k \, g(\mathbf{X}_k, \mathbf{U}_k, \tau_k). \tag{1.18}$$

The boundary and path constraints are approximated as

$$\boldsymbol{\phi}(\mathbf{X}_0, \mathbf{X}_f) = \mathbf{0} \tag{1.19}$$

$$\boldsymbol{\Gamma}(\mathbf{X}_k, \mathbf{U}_k, \tau_k) \leq \mathbf{0} \ (k = 1, 2, ..., N), \tag{1.20}$$

respectively. The cost (1.18) must be minimized subject to the constraints (1.15), (1.16), (1.19), and (1.20). This is a nonlinear programming (NLP) problem, which can be solved with any number of established software packages. In this thesis, we used the `fmincon` function in MATLAB®, version 9.7 R2019b.

Chapter 2

Optimal Control Problem for 3/4 CMG Configuration

2.1 Optimal Control Formulation

In this chapter, we apply the Gauss pseudospectral method to solve for gimbal angle trajectories in the 3/4 CMG configuration. For our spacecraft maneuver using the nonredundant 3/4 pyramid configuration, we take the state $\mathbf{x}$ to be

$$\mathbf{x} = \begin{bmatrix} \mathbf{q} \\ \boldsymbol{\omega} \\ \boldsymbol{\delta} \end{bmatrix} \in \mathbb{R}^{10}. \tag{2.1}$$

The control will be taken as $\mathbf{u} = \dot{\boldsymbol{\delta}} \in \mathbb{R}^3$. From equations 1.1 and 1.6, the states and controls must satisfy the coupled differential equations

$$\dot{\mathbf{q}}_v = -\tfrac{1}{2}\boldsymbol{\omega} \times \mathbf{q}_v + \tfrac{1}{2}q_4\boldsymbol{\omega} \tag{2.2a}$$

$$\dot{q}_4 = -\tfrac{1}{2}\boldsymbol{\omega} \cdot \mathbf{q}_v \tag{2.2b}$$

$$\dot{\boldsymbol{\omega}} = -J^{-1}\Big[\boldsymbol{\omega} \times \big(J\boldsymbol{\omega} + \boldsymbol{h}\big) + A\mathbf{u}\Big] \tag{2.3}$$

$$\dot{\boldsymbol{\delta}} = \mathbf{u}. \tag{2.4}$$

The initial time will be fixed at $t_0 = 0$. The final time t_f will be free. Further constraints such as maximum spacecraft slew rate or maximum gimbal rate may also be specified.

In the next section, we will present the computation results given the specified parameters of the problem. We choose a preliminary cost function $C_1 = t_f$ so as to purely optimize maneuver time. However, it is desirable to have a maneuver that maximally avoids singularity. We then use the calculated value of t_f for a second calculation. In the second calculation, we set an upper bound on the final time ($\sim 1.35t_f$) and use the cost function

$$C_2 = -\int_0^{t_f} \big|\det(A)\big|\, dt. \tag{2.5}$$

By minimizing C_2, we essentially are maximizing the integrated distance from singularity. Results are presented for scenarios where the gimbal angles initially start in both nonsingular and singular positions.

2.2 Simulation Results

It is important to review the assumptions made in our maneuver problem:

1. There are no external torques on the spacecraft.
2. Each CMG's gimbal rate $\dot{\delta}_i$ is negligible compared to its rotor angular velocity.
3. The spacecraft inertia tensor J is approximated as constant, even though it technically varies according to the orientations of the CMG rotors (see Appendix A).
4. Each CMG rotor's center of mass is fixed within the spacecraft at all times.

5. Each CMG rotor's axial direction is a principal axis of the rotor.

Note that we made no assumption that the gyroscopic term $\boldsymbol{\omega} \times \left(J\boldsymbol{\omega} + \boldsymbol{h}\right)$ is negligible.

We assumed a 3/4 pyramid configuration with a skew angle of $\beta = 53.13°$ (i.e. $\cos\beta = 0.6$). We simulated a rest-to-rest 47° right-handed roll about the x-axis (this example was chosen so as to be able to compare with the results of the example from [1]). The simulation parameters are summarized in Table 2.1.

Table 2.1: Simulation parameters for 3/4 CMG configuration example

Skew angle, β	$53.13°$
CMG axial momentum, η	$1000\ \mathrm{N \cdot m \cdot s}$
Spacecraft inertia tensor, J	$\mathrm{diag}(21400, 20100, 5000)\ \mathrm{kg \cdot m^2}$
Max gimbal angle rate	$2\ \mathrm{rad/s}$
Max spacecraft slew rate	$10\ \mathrm{deg/s}$
Initial angular velocity, $\boldsymbol{\omega}(0)$	$(0, 0, 0)^T$
Final angular velocity, $\boldsymbol{\omega}(t_f)$	$(0, 0, 0)^T$
Initial quaternion, $\mathbf{q}(0)$	$(0, 0, 0, 1)^T$
Final quaternion, $\mathbf{q}(t_f)$	$(0.4,\ 0,\ 0,\ \sqrt{0.84})^T$
Initial gimbal angles (Case 1)	$(60,\ 180,\ -60)^T$ deg.
Initial gimbal angles (Case 2)	$(90,\ 0,\ -90)^T$ deg.

We considered two cases. In Case 1, the initial gimbal angles were chosen as $\boldsymbol{\delta}(0) = (60,\ 180,\ -60)^T$ deg. These angles are nonsingular "preferred angles," calculated using the method demonstrated by Vadali et al. [5]. In Case 2, we chose the initial gimbal angles as $\boldsymbol{\delta}(0) = (90,\ 0,\ -90)^T$ deg. This is a singular position, where $\det(A) = 0$.

We proceeded in two stages. In the first stage, we computed the maneuver using the simple cost function $C_1 = t_f$, so as to purely minimize maneuver time. In the second stage, we used the computed value of t_f to set an upper bound on maneuver time

($\sim 1.35t_f$) and then re-calculated the maneuver using a new cost function C_2, as given in equation 2.5. Of course, if an estimate on the maneuver time is already known, one can skip directly to the second stage. (The computation of the second stage is generally faster.) All computations were done with $N = 19$ nodes (20 total nodes including $\tau_0 = -1$).

Results from Case 1 are plotted in Figures 2.1 and 2.2. Results from Case 2 are plotted in Figures 2.3 and 2.4. We may normalize the Jacobian by defining $\tilde{A} \doteq A/\eta$. The normalized Jacobian determinants are plotted in Figure 2.5. The maneuver trajectories were plotted in comparison with the trajectories computed using the generalized singularity robust (SR) steering [1] applied to the (non-failed) 4-CMG pyramid configuration. Note that the generalized SR steering performs an eigenaxis rotation, not necessarily with optimization. For being in a failure state, the maneuver computed by the Gauss pseudospectral method performs reasonably well when compared to the generalized SR non-failure performance; the generalized SR maneuver takes 8 seconds, and the Gauss PS maneuver takes 12 seconds, while still maximally avoiding singularity. We see in the determinant plots in Figure 2.5 that the CMG configuration inevitably passes through singularities, but it is able to do so quickly and without lingering at the singularities. We also verified that both the magnitude of total system angular momentum, $|J\boldsymbol{\omega} + \boldsymbol{h}|$, is constant and the quaternion magnitude is constant at 1, but we omitted the plots for brevity.

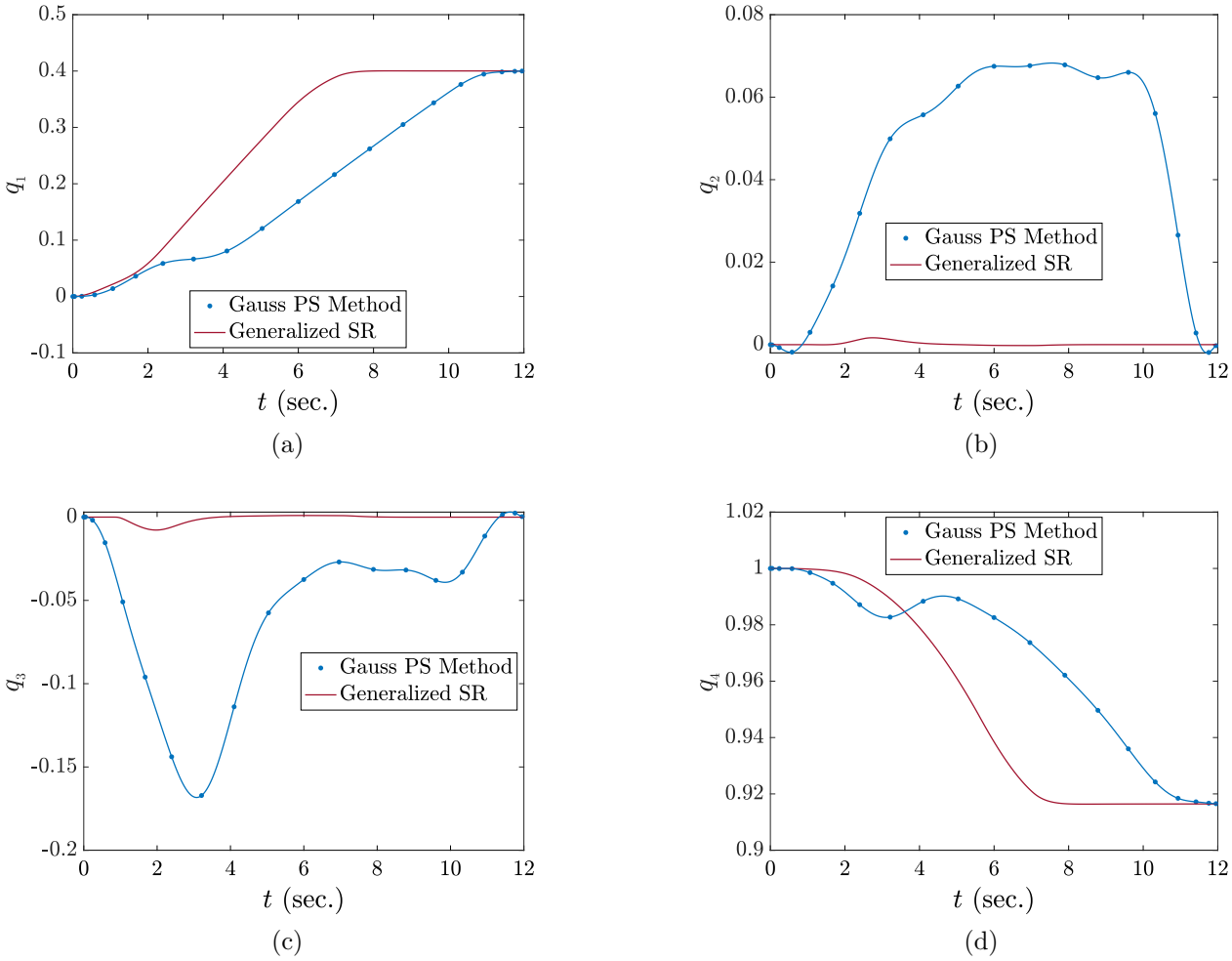


Fig. 2.1: Quaternion components comparison between Gauss PS applied to 3/4 configuration and Generalized SR applied to non-failure configuration (Case 1)

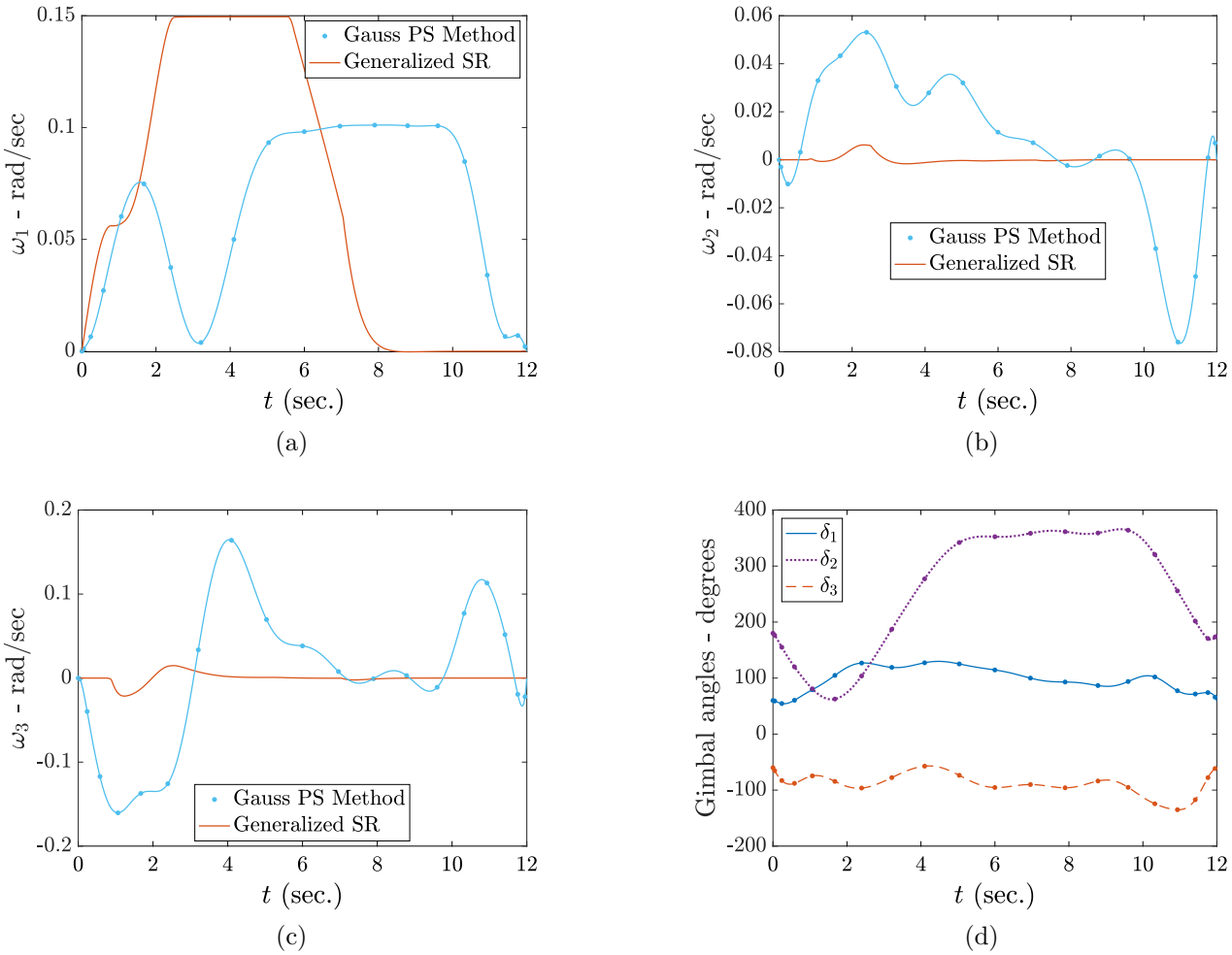


Fig. 2.2: Angular velocity components and gimbal angles in the 3/4 CMG configuration (Case 1). (a-c) Angular velocity components of the 3/4 configuration, obtained via the Gauss PS method, are compared with a baseline non-failure configuration obtained via the Generalized SR method. (d) The corresponding gimbal angle evolution for the 3/4 configuration.

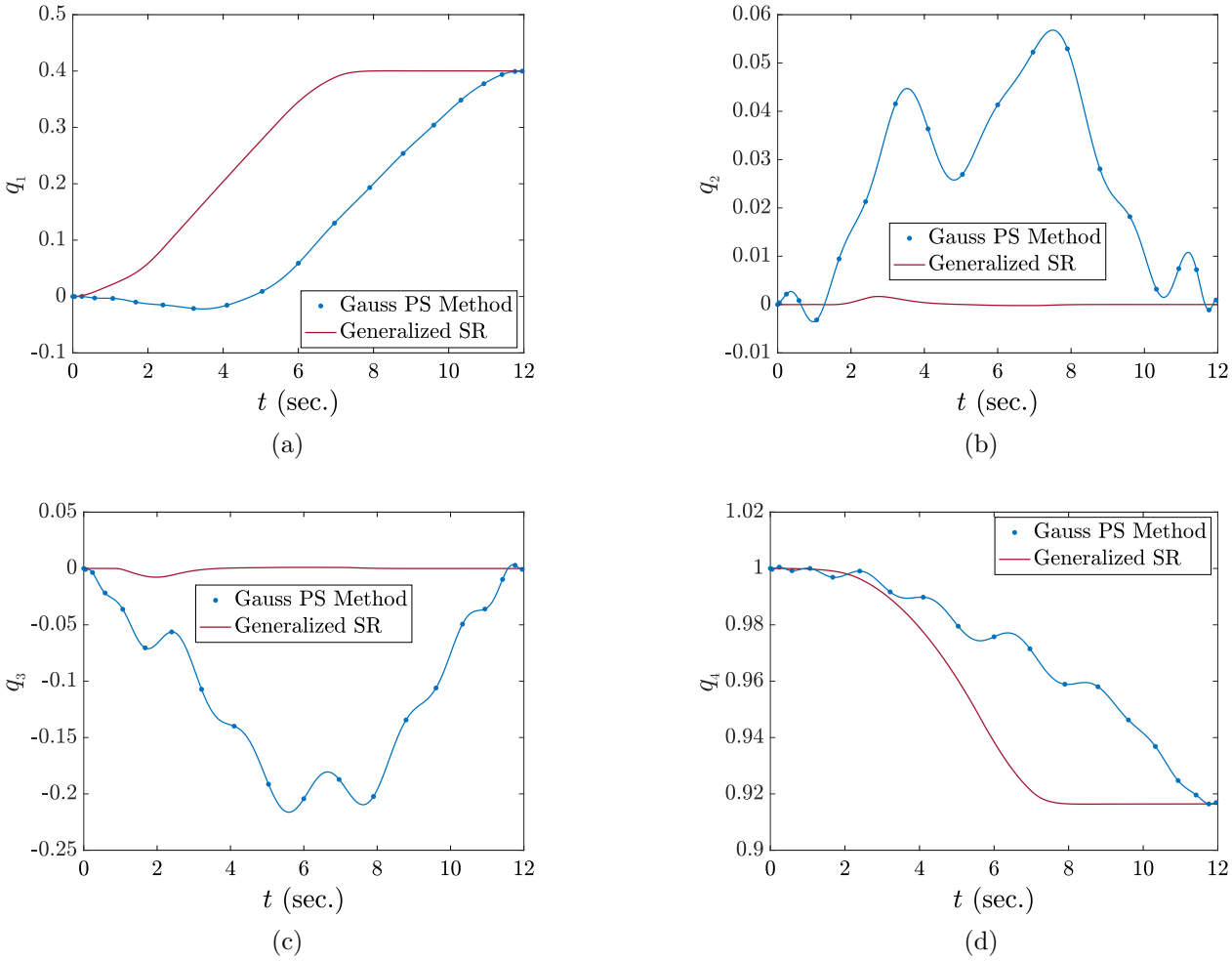


Fig. 2.3: Quaternion components comparison between Gauss PS applied to 3/4 configuration and Generalized SR applied to non-failure configuration (Case 2)

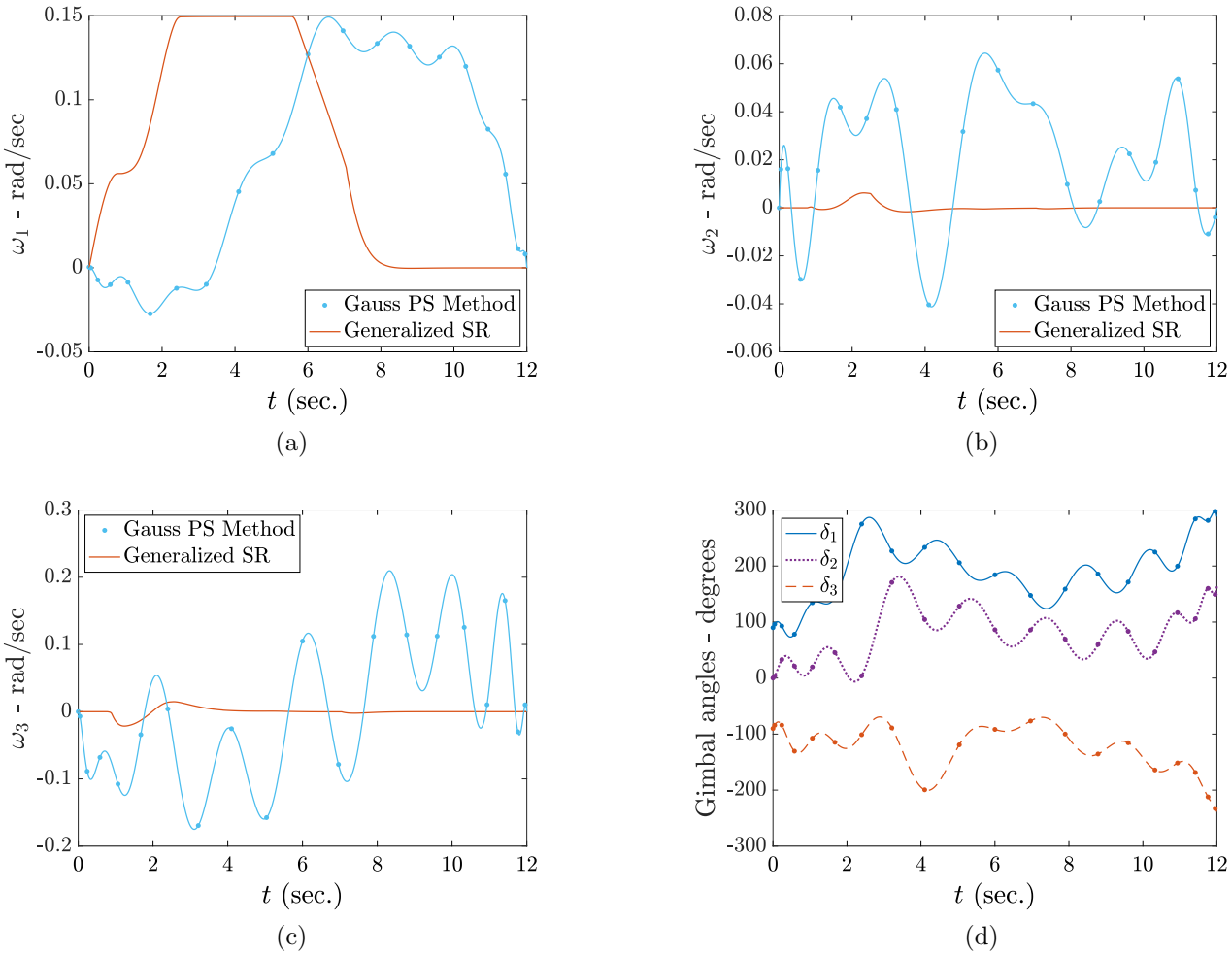
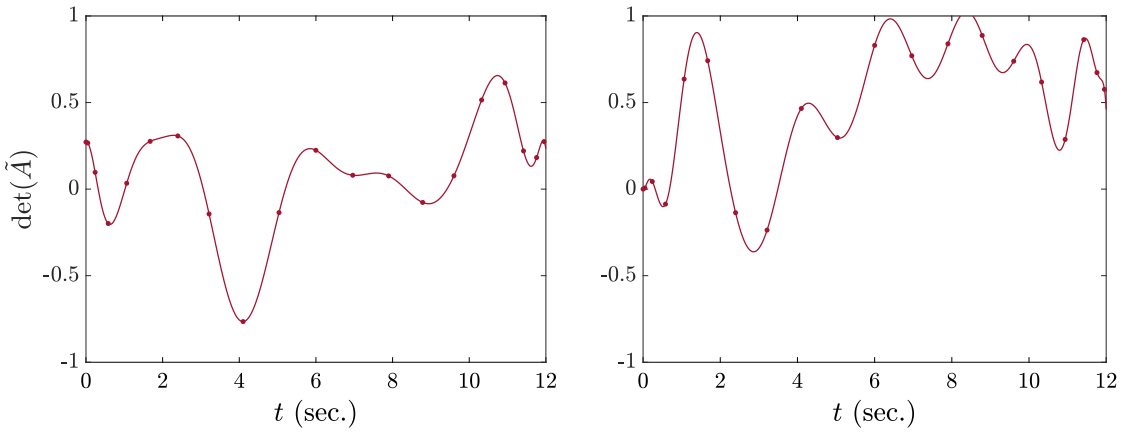


Fig. 2.4: Angular velocity components and gimbal angles in the 3/4 CMG configuration (Case 2). (a-c) Angular velocity components of the 3/4 configuration, obtained via the Gauss PS method, are compared with a baseline non-failure configuration obtained via the Generalized SR method. (d) The corresponding gimbal angle evolution for the 3/4 configuration.



(a) Case 1 (b) Case 2

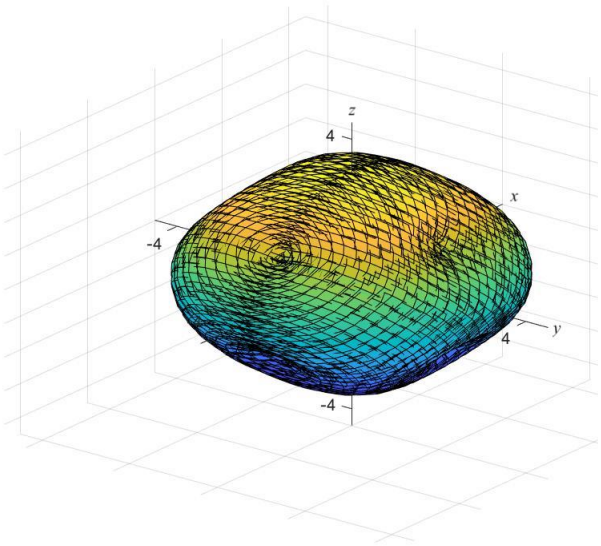
Fig. 2.5: Normalized Jacobian determinant from Gauss pseudospectral maneuver (both cases)

Chapter 3

Reduction of CMG Momentum

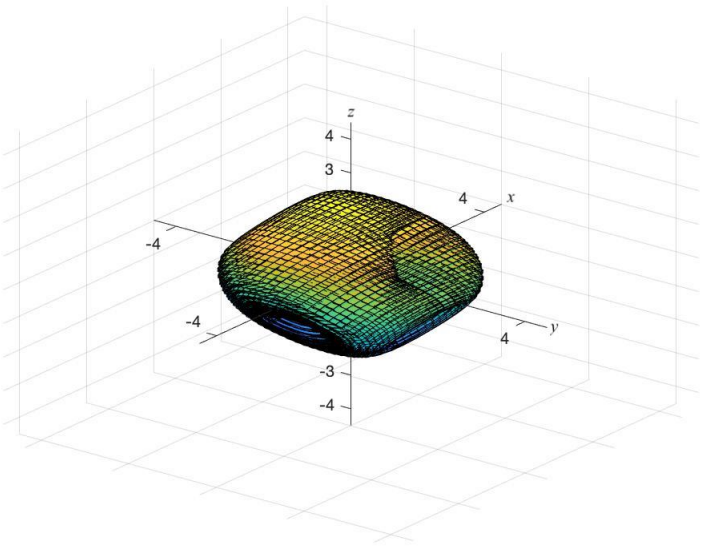
3.1 Necessity for Angular Momentum Reduction

In Chapter 4, we will investigate the possibility of applying the Gauss pseudospectral method to solve for solutions in the case of the 2/4 CMG pyramid configuration. However, with too few CMGs available, the angular momentum envelope of the CMG system becomes smaller, severely deflated, and highly non-spherical. See Figure 3.1. (There are two possible cases in the 2/4 pyramid configuration: either the two CMGs are adjacent or they are opposite of each other.) This leads to elevated chances that, from any given initial set of gimbal angles, a desired spacecraft rotation will be physically impossible.



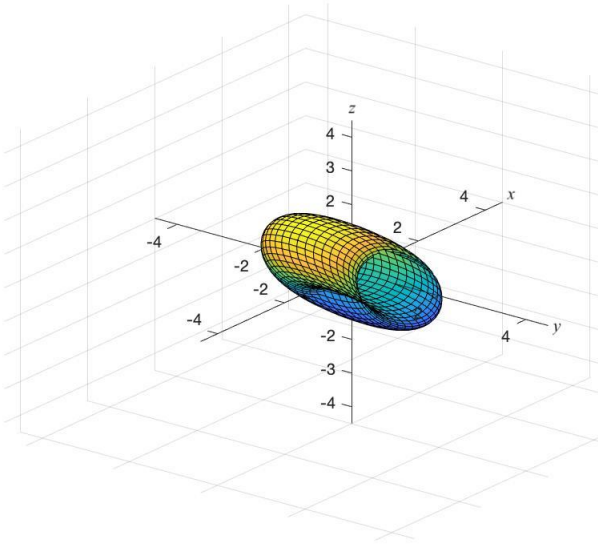
(a)

4-CMG



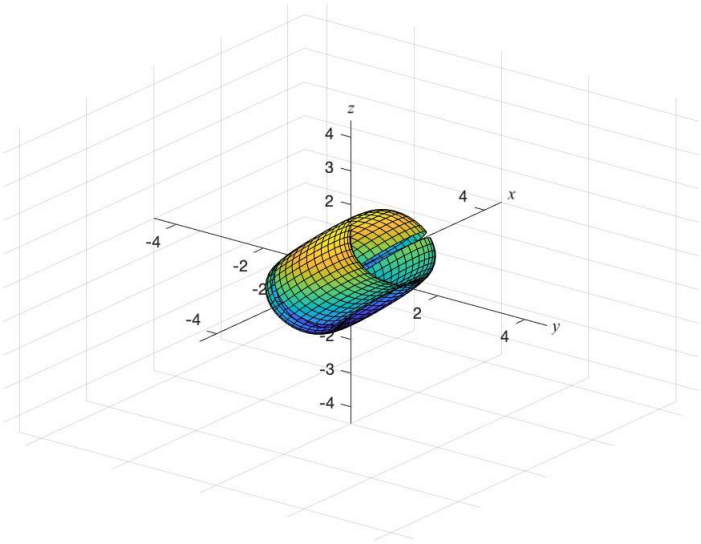
(b)

3-CMG



(c)

2-CMG (adjacent rotors)



(d)

2-CMG (opposite rotors)

Fig. 3.1: Angular momentum envelopes for various CMG configurations

For example, consider a spacecraft with a 2/4 CMG configuration of adjacent rotors. Let the initial gimbal angles be $[\delta_1,\ \delta_2] = [90°,\ 90°]$ (this corresponds to maximum angular momentum in the z-direction). The corresponding angular momentum envelope is shown in Figure 3.2.

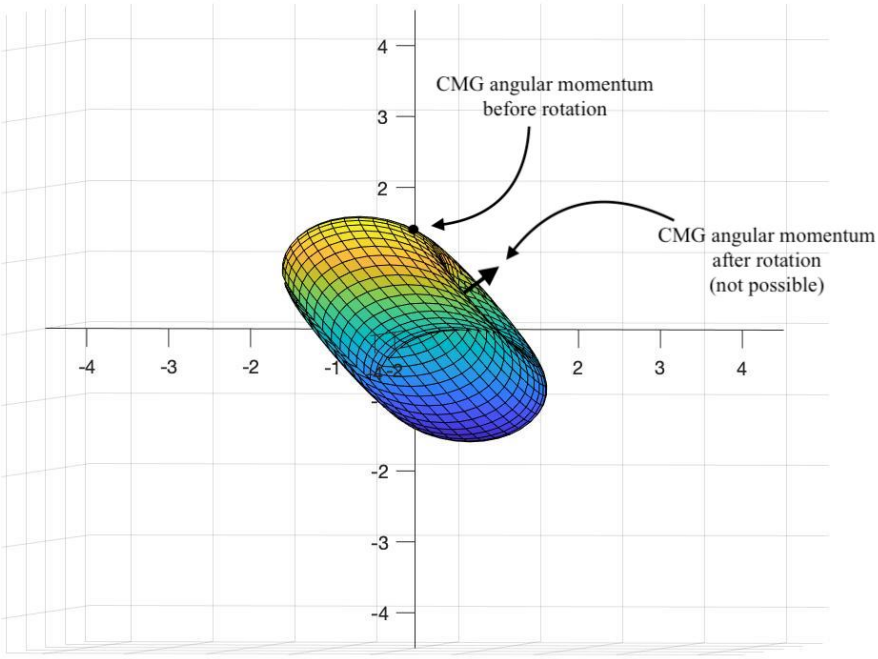


Fig. 3.2: Example showing impossible spacecraft rotation

If the desired maneuver is a 45° left-handed roll about the y-axis, the angular momentum vector after the rotation would lie outside the momentum envelope. **This maneuver is impossible.** Thus, we would need an external torque (e.g. magnetic torquers or thrusters) to essentially "beat down" this angular momentum vector into its envelope.

In the next section, we will discuss a strategy for using an external torque to desaturate the CMG angular momentum and bring the momentum vector close to zero, away from the edge of the envelope.

3.2 Strategy for Angular Momentum Reduction

We wish to make use of an external torque (such as magnetic torquers or thrusters) to reduce a saturated CMG momentum to be closer to zero. Specifically, given some current gimbal angles $\boldsymbol{\delta}$, we wish to command the gimbal angles to some final values $\boldsymbol{\delta}_f$. First, we need to define some saturation functions.

Given a scalar x and a positive constant L, define the *scalar saturation function* as

$$\underset{L}{\text{sat}}(x) = \begin{cases} L & \text{if} \quad x \geq L \\ x & \text{if} \quad |x| \leq L \\ -L & \text{if} \quad x \leq -L \end{cases} \cdot$$

We can apply the scalar saturation function component-wise to a vector $\mathbf{x} = (x_1, x_2, ..., x_n)^T$,

$$\underset{L_i}{\text{sat}}(\mathbf{x}) = \begin{bmatrix} \text{sat}_{L_1}(x_1) \\ \text{sat}_{L_2}(x_2) \\ \vdots \\ \text{sat}_{L_n}(x_n) \end{bmatrix},$$

given the positive constants $L_1, L_2, ..., L_n$.

Now given a vector $\mathbf{x}$ and a positive constant U, define the *vector saturation function* as

$$\underset{\mu,U}{\overrightarrow{\text{sat}}}(\mathbf{x}) = \begin{cases} \mathbf{x} & \text{if} \quad \mu(\mathbf{x}) < U \\ \dfrac{U\mathbf{x}}{\mu(\mathbf{x})} & \text{if} \quad \mu(\mathbf{x}) \geq U \end{cases} .$$

Here, $\mu(\mathbf{x})$ is a positive scalar norm of $\mathbf{x}$, which is chosen in context. For example, we may choose the L-2 norm $\mu(\mathbf{x}) = \|\mathbf{x}\|_2 = \sqrt{\mathbf{x} \cdot \mathbf{x}}$ or the L-∞ norm $\mu(\mathbf{x}) = \|\mathbf{x}\|_\infty =$

$\max\{|x_1|, ..., |x_n|\}$.

We are ready to describe the developed steering logic for reduction of CMG angular momentum. We need an external torque $\boldsymbol{\tau}_{c,\text{ext}}$ in order to reduce the CMG configuration angular momentum. Following the quaternion feedback logic developed in [1, 7], we propose a steering law for reduction of the CMG angular momentum, given each max external torque $\tau_{\max,x} = \tau_{\max,y} = \tau_{\max,z} = \tau_{\max}$,

$$\boldsymbol{\tau} = -J\left[2k \underset{L_i}{\text{sat}}\left(\boldsymbol{e} + \frac{1}{T}\int \boldsymbol{e}\right) + c\boldsymbol{\omega}\right] + \boldsymbol{\omega} \times \boldsymbol{h} + A\dot{\boldsymbol{\delta}} \tag{3.1}$$

$$L_i = \frac{c}{2k}\,\min\left\{\sqrt{4a_i|e_i|},\ |\omega_i|_{\max}\right\} \tag{3.2}$$

$$\boldsymbol{\tau}_{c,\text{ext}} = \underset{\tau_{\max,i}}{\text{sat}}\left(\boldsymbol{\tau}\right) \tag{3.3}$$

$$\dot{\boldsymbol{\delta}}_d = -\frac{1}{T_{\text{com}}}(\boldsymbol{\delta} - \boldsymbol{\delta}_f) \tag{3.4}$$

$$\dot{\boldsymbol{\delta}}_c = \underset{\mu,U}{\overrightarrow{\text{sat}}}(\dot{\boldsymbol{\delta}}_d),\ \ \mu(\dot{\boldsymbol{\delta}}_d) = \|A\dot{\boldsymbol{\delta}}_d\|_\infty,\ \ U = 0.7\tau_{\max}. \tag{3.5}$$

Here, $\mathbf{e}$ is the quaternion error vector, given by $\mathbf{e} = (e_1, e_2, e_3)$, where (e_1, e_2, e_3, e_4) is the error quaternion, given by

$$\begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \end{bmatrix} = \begin{bmatrix} q_{4c} & q_{3c} & -q_{2c} & -q_{1c} \\ -q_{3c} & q_{4c} & q_{1c} & -q_{2c} \\ q_{2c} & -q_{1c} & q_{4c} & -q_{3c} \\ q_{1c} & q_{2c} & q_{3c} & q_{4c} \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix}$$

where (q_1, q_2, q_3, q_4) is the spacecraft's current quaternion and $(q_{1c}, q_{2c}, q_{3c}, q_{4c})$ is the

commanded quaternion, i.e. the desired final quaternion $\mathbf{q}(t_f)$. We choose $k = 9.54$, $c = 5.5$, and a_i as 40% the maximum angular acceleration in the ith direction [1]

$$a_i = 0.4\frac{\tau_{\max}}{J_{ii}}.$$

Essentially, the goal of this strategy is to allow the gimbal angles to approach new commanded final gimbal angles $\boldsymbol{\delta}_f$ based on a desired time constant T_{com}. In changing the gimbal angles, an external torque $\boldsymbol{\tau}$ must be applied to counteract the effective torque due to the changing gimbal angles, $A\dot{\boldsymbol{\delta}}$ (provided in equation 3.1), while also keeping the spacecraft pointing in a desired direction, implicit in $\mathbf{e}$. The hardware limit of max torque $\tau_{\max}$ is accounted for in equations 3.3 and 3.5. Equation 3.5 ensures that the gimbal angles don't change so fast so that the external torque is incapable of controlling the vehicle. The resulting commanded torque $\boldsymbol{\tau}_{c,\text{ext}}$ and commanded gimbal rates $\dot{\boldsymbol{\delta}}_c$ are then executed by the spacecraft.

The next section exhibits an example using this momentum reduction strategy.

3.3 Example using Angular Momentum Reduction

We provide an example where two adjacent CMGs are in failure and only two CMGs are functioning. Assume the spacecraft has a 2/4 adjacent CMG configuration. Without loss of generality, let the functioning CMGs be #1 and #2. Suppose the two CMGs are saturated in momentum, so that their initial gimbal angles are $[\delta_1,\ \delta_2] = [90°,\ 90°]$ (this corresponds to maximum angular momentum in the z-direction). To desaturate and bring the CMG angular momentum close to zero, we wish to transition the gimbal angles to $[\delta_1,\ \delta_2] = [90°,\ -90°]$. Finally, we'll start the spacecraft rotated 10° off the nominal pointing target, to demonstrate the ability to stay aligned in desired attitude.

The simulation parameters are summarized in Table 3.1.

Table 3.1: Simulation parameters in momentum reduction example

Functioning CMGs	#1, #2
CMGs in failure	#3, #4
Skew angle, β	53.13°
CMG axial momentum, η	$1000\ \mathrm{N \cdot m \cdot s}$
Spacecraft inertia tensor, J	$\mathrm{diag}(21400, 20100, 5000)\ \mathrm{kg \cdot m^2}$
Max gimbal angle rate	2 rad/s
Max spacecraft slew rate	10 deg/s
Initial angular velocity, $\boldsymbol{\omega}(0)$	$(0, 0, 0)^T$
Final angular velocity, $\boldsymbol{\omega}(t_f)$	$(0, 0, 0)^T$
Initial quaternion, $\mathbf{q}(0)$	$(\sin 5^\circ, 0, 0, \cos 5^\circ)^T$
Final quaternion, $\mathbf{q}(t_f)$	$(0, 0, 0, 1)^T$
Commanded time-constant, T_{com}	5 seconds
Max external torque, $\boldsymbol{\tau}_{\mathrm{max}}$	$100\ \mathrm{N \cdot m}$
Initial gimbal angles	$(90, 90)^T$ deg.
Final gimbal angles	$(90, -90)^T$ deg.

The simulation results are shown in Figures 3.3 and 3.4.

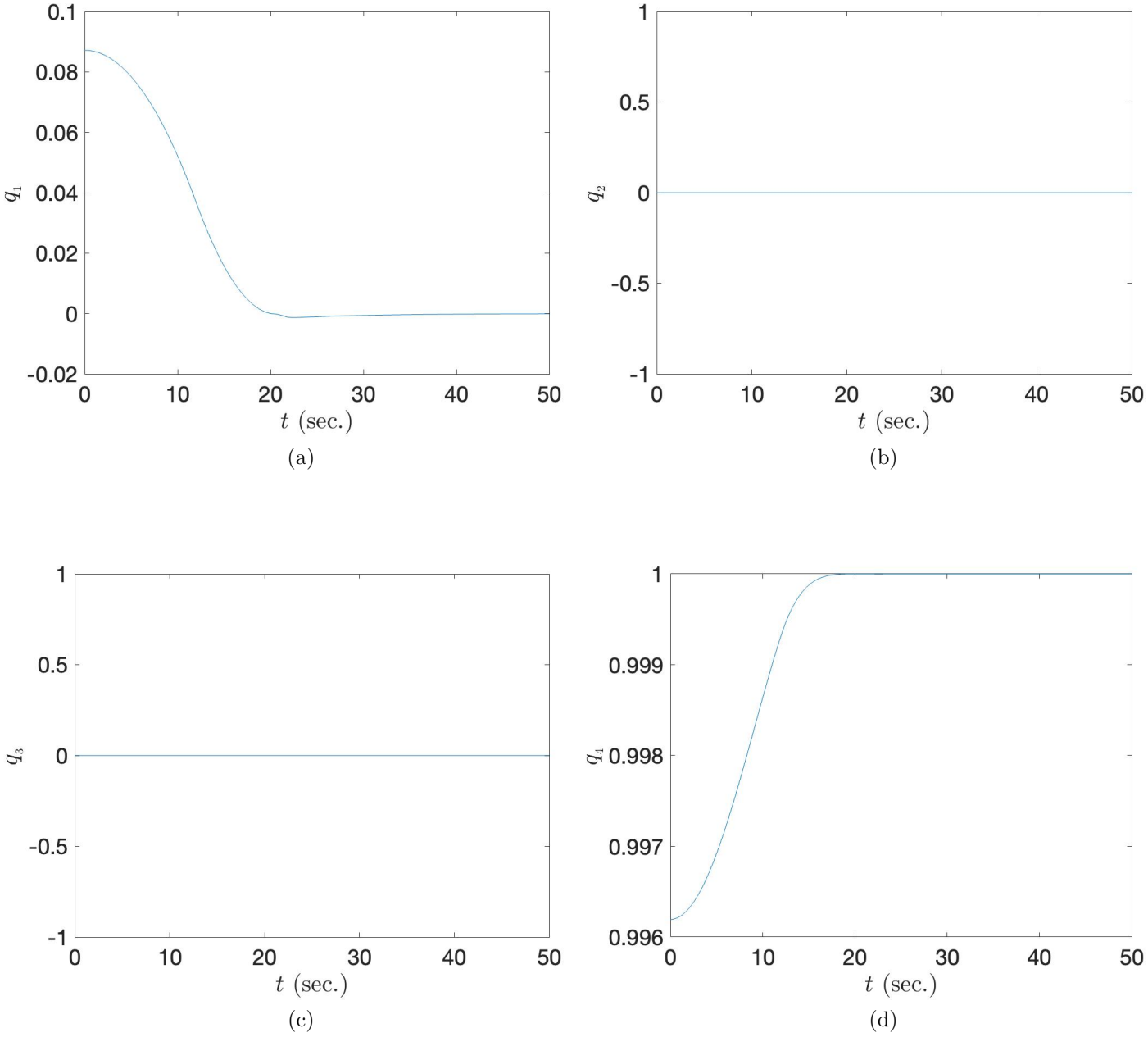


Fig. 3.3: Quaternion components in momentum reduction example

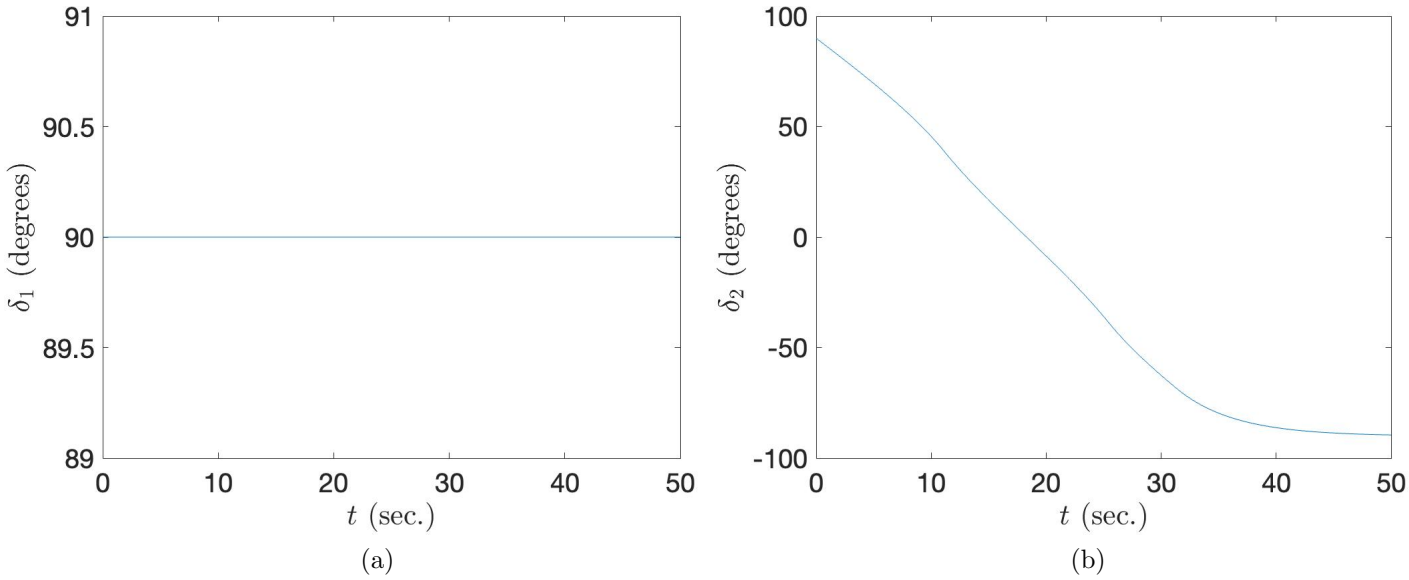


Fig. 3.4: Gimbal angles in momentum reduction example

We see in this example that the gimbal angles can transition to the desired values, desaturating the CMG momentum, while bringing the spacecraft attitude back to the nominal quaternion $(0, 0, 0, 1)$.

In the next chapter, we assume the CMG momentum is already desaturated and then make use of the Gauss pseudospectral method to compute maneuvers for the 2/4 CMG configurations.

Chapter 4

Optimal Control Problem for 2/4 CMG Configuration

4.1 Adjacent Configuration

With 2/4 CMGs functioning in the pyramid configuration, there are two possibilities: the two functioning CMGs are adjacent to each other or they are opposite to each other. This section will show simulation results of the former, and the next section will show the latter.

With the methods introduced in the previous chapter, we know that the CMG angular momentum can be desaturated to be close to zero with an external torque. In the examples of this chapter, we will assume this has already been done. By first bringing the CMG total momentum close to zero, we ensure that a solution exists for a given desired spacecraft rotation. The Gauss pseudospectral method can then be used to compute the attitude and gimbal trajectories to perform a desired maneuver.

We present an example where the Gauss pseudospectral method was used to compute gimbal angle trajectories for a spacecraft rotation in the case of a 2/4 adjacent CMG configuration. The same assumptions as in Section 2.2 were made. With two CMGs, the Jacobian always has rank less than 3, so it makes no sense to avoid singularity in the 2/4 configuration. Thus our cost function was chosen as $C = t_f$ so as to minimize the time for maneuver. The simulation parameters are summarized in the following table.

Table 4.1: Simulation parameters (2/4 adjacent configuration)

Functioning CMGs	#1, #2
CMGs in failure	#3, #4
Skew angle, β	53.13°
CMG axial momentum, η	$1000\ \mathrm{N \cdot m \cdot s}$
Spacecraft inertia tensor, J	$\mathrm{diag}(21400, 20100, 5000)\ \mathrm{kg \cdot m^2}$
Max gimbal angle rate	2 rad/s
Max spacecraft slew rate	10 deg/s
Initial angular velocity, $\boldsymbol{\omega}(0)$	$(0, 0, 0)^T$
Final angular velocity, $\boldsymbol{\omega}(t_f)$	$(0, 0, 0)^T$
Initial quaternion, $\mathbf{q}(0)$	$(0, 0, 0, 1)^T$
Final quaternion, $\mathbf{q}(t_f)$	$(0.4,\ 0,\ 0,\ \sqrt{0.84})^T$
Initial gimbal angles	$(90,\ -90)^T$ deg.

The maneuver performed in this simulation corresponds to a 47° right-handed roll about the x-axis. The gimbal angles begin at the near-zero-momentum state of $(90,\ -90)$ degrees. The simulation results are shown below. Again, the trajectories are compared with those computed using the generalized singularity robust (SR) method [1] applied to the non-failure CMG pyramid configuration.

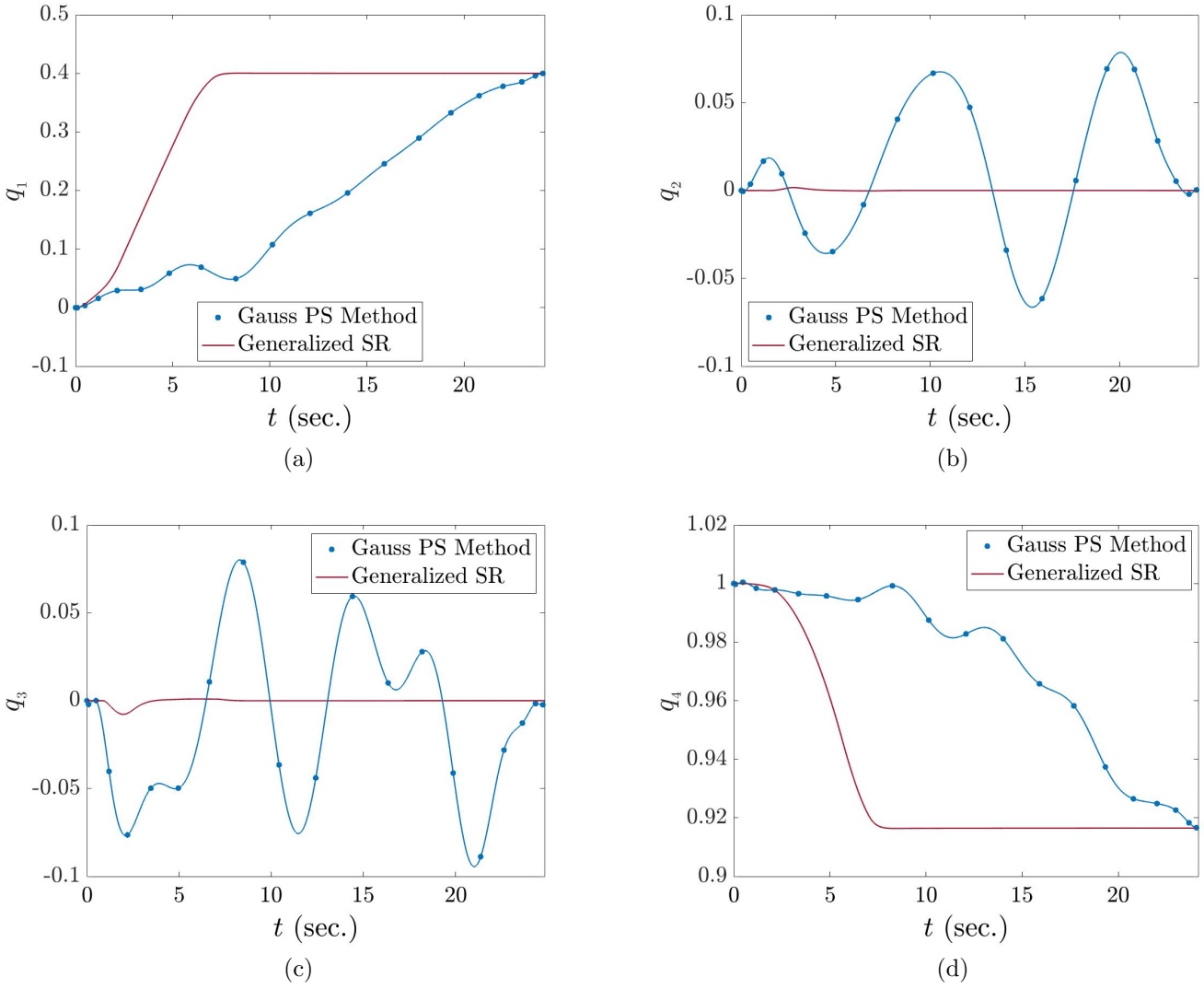


Fig. 4.1: Quaternion components comparison between Gauss PS applied to 2/4 configuration and Generalized SR applied to non-failure configuration (2/4 adjacent case)

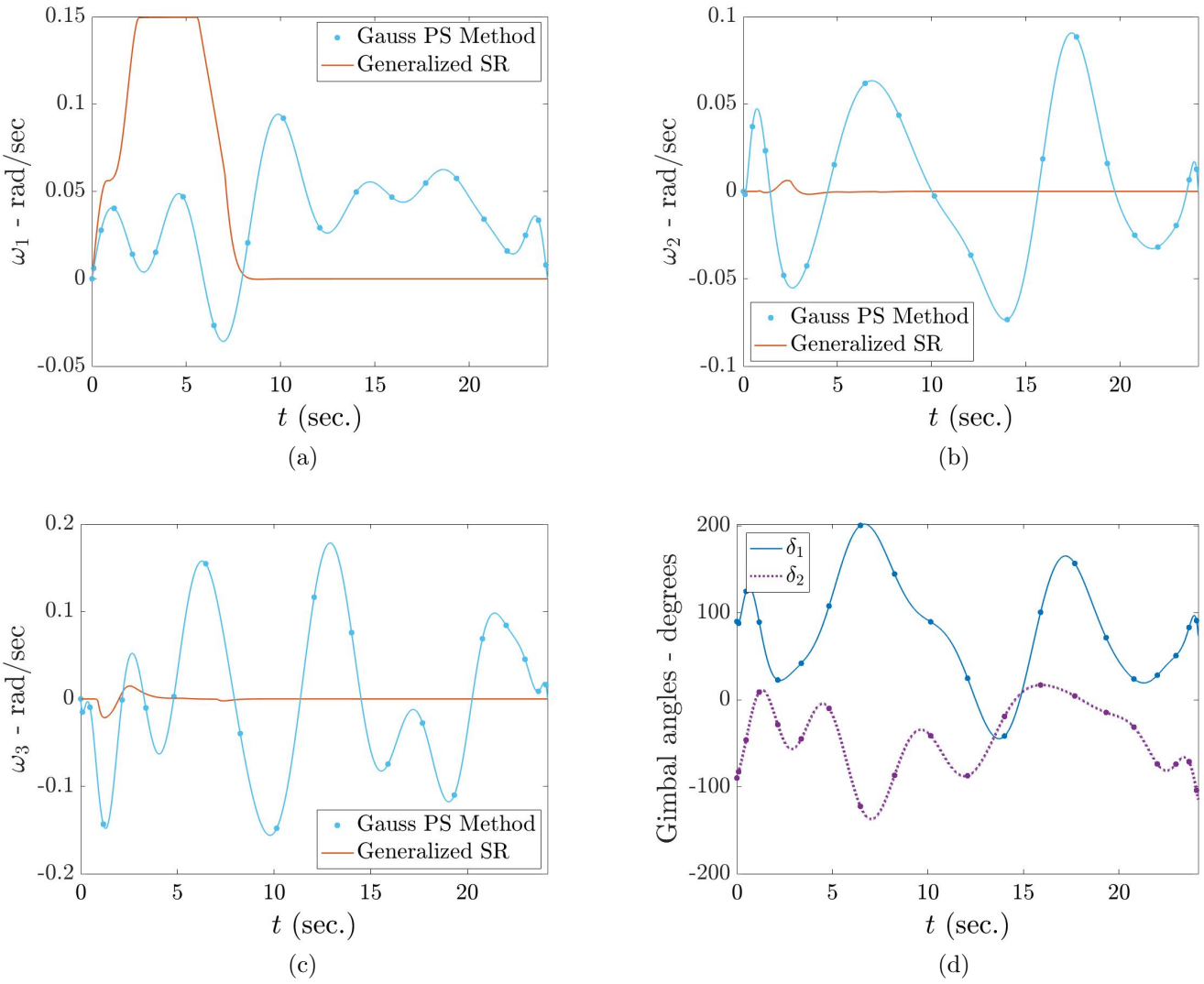


Fig. 4.2: Angular velocity components and gimbal angles in the 2/4 adjacent CMG configuration. (a-c) Angular velocity components of the 2/4 configuration, obtained via the Gauss PS method, are compared with a baseline non-failure configuration obtained via the Generalized SR method. (d) The corresponding gimbal angle evolution for the 2/4 configuration.

It has also been verified that the total system angular momentum is constant, and the quaternion magnitude is constant at 1, but the plots are omitted for brevity.

For being in a severe failure case, the 2/4 CMG configuration with the Gauss pseudospectral method performs reasonably well compared to the non-failure generalized SR method.

4.2 Opposite Configuration

This section will show simulation results of the 2/4 opposite CMG configuration. Again, we will assume that the total CMG momentum has been brought to be close to zero. The same assumptions as in Section 2.2 were made, and our cost function was chosen as $C = t_f$ so as to minimize the time for maneuver. The simulation parameters are summarized in the following table.

Table 4.2: Simulation parameters (2/4 opposite configuration)

Functioning CMGs	#1, #3
CMGs in failure	#2, #4
Skew angle, β	53.13°
CMG axial momentum, η	$1000\ \mathrm{N \cdot m \cdot s}$
Spacecraft inertia tensor, J	$\mathrm{diag}(21400, 20100, 5000)\ \mathrm{kg \cdot m^2}$
Max gimbal angle rate	2 rad/s
Max spacecraft slew rate	10 deg/s
Initial angular velocity, $\boldsymbol{\omega}(0)$	$(0, 0, 0)^T$
Final angular velocity, $\boldsymbol{\omega}(t_f)$	$(0, 0, 0)^T$
Initial quaternion, $\mathbf{q}(0)$	$(0, 0, 0, 1)^T$
Final quaternion, $\mathbf{q}(t_f)$	$(0.4,\ 0,\ 0,\ \sqrt{0.84})^T$
Initial gimbal angles	$(10,\ -10)^T$ deg.

The maneuver performed in this simulation corresponds to a 47° right-handed roll about the x-axis. The gimbal angles begin at the near-zero-momentum state of $(10,\ -10)$ degrees. The simulation results are shown below. Again, the trajectories are compared with those computed using the generalized singularity robust (SR) method [1] applied to the non-failure CMG pyramid configuration.

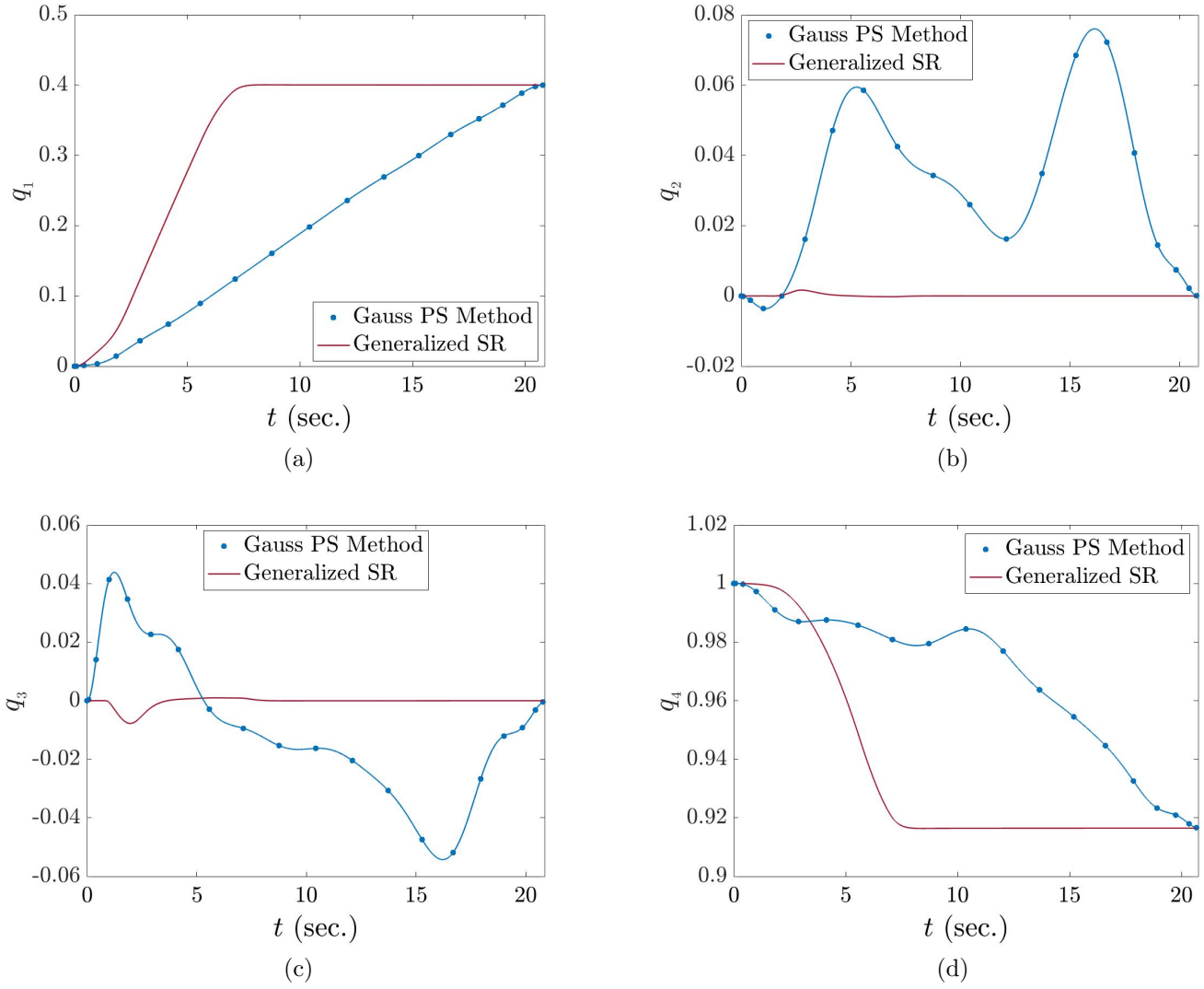


Fig. 4.3: Quaternion components comparison between Gauss PS applied to 2/4 configuration and Generalized SR applied to non-failure configuration (2/4 opposite case)

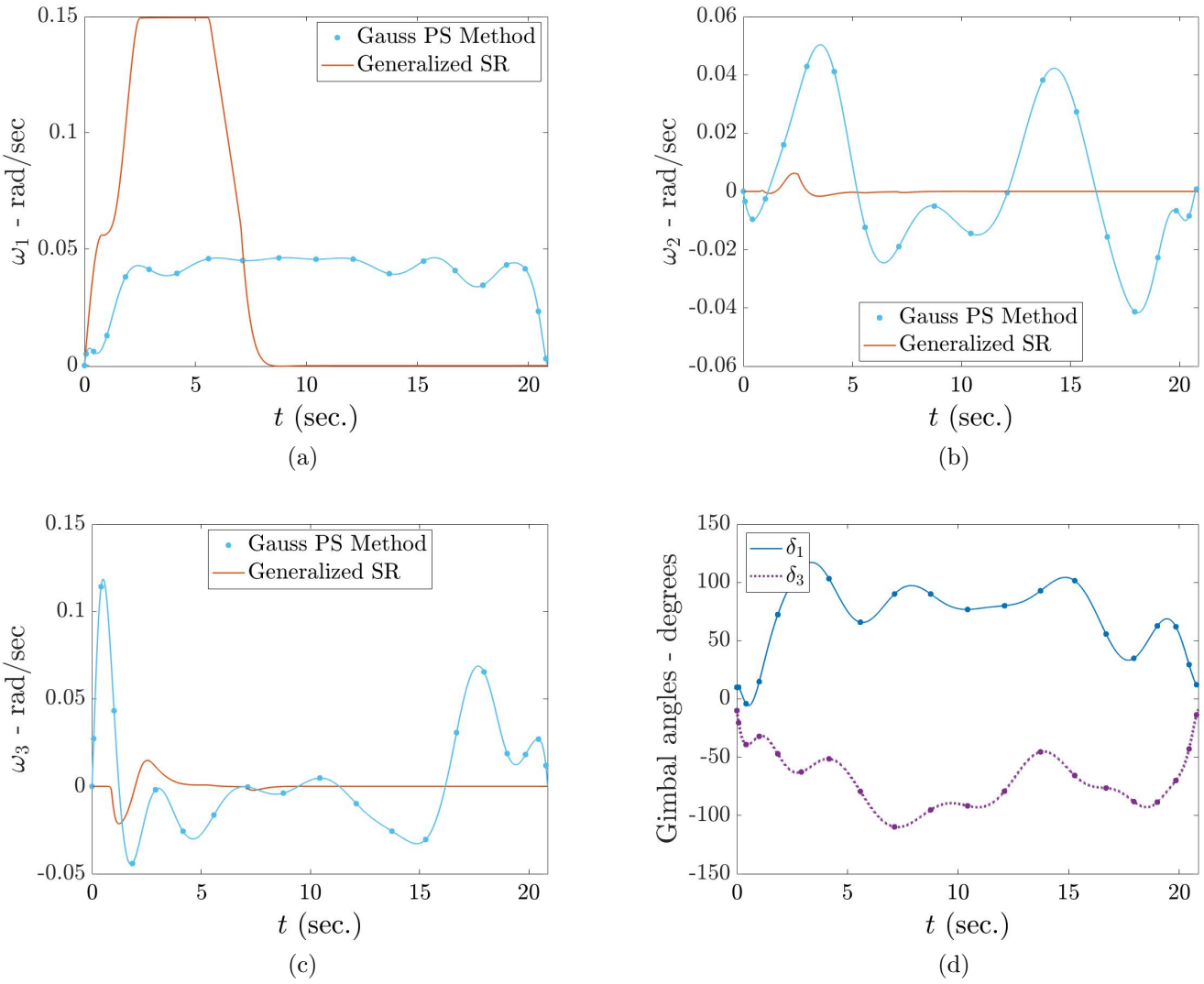


Fig. 4.4: Angular velocity components and gimbal angles in the 2/4 opposite CMG configuration. (a-c) Angular velocity components of the 2/4 configuration, obtained via the Gauss PS method, are compared with a baseline non-failure configuration obtained via the Generalized SR method. (d) The corresponding gimbal angle evolution for the 2/4 configuration.

It has also been verified that the total system angular momentum is constant, and the quaternion magnitude is constant at 1, but the plots are omitted for brevity.

For being in a severe failure case, the 2/4 CMG configuration with the Gauss pseudospectral method performs reasonably well compared to the non-failure generalized SR method.

Chapter 5

Conclusion

In this thesis, the Gauss pseudospectral method was successfully applied to computing gimbal angle trajectories of the 3/4 CMG pyramid configuration and the 2/4 CMG pyramid configuration. Since the 3/4 pyramid configuration is nonredundant, care was taken to maximally avoid singularities. From the computation results, the maneuver fared reasonably well compared to the non-failure performance of the generalized singularity robust steering.

In the case of the 2/4 CMG configuration, an additional step was required to precondition the gimbal angles. If the total CMG momentum is too close to the edge of its momentum envelope, a desired spacecraft slew may be theoretically impossible. An external torque is first required to reduce the CMG momentum to be close to zero. A quaternion-feedback steering law was developed to use an external torque to desaturate the total CMG angular momentum. Finally, the Gauss pseudospectral method was used to then compute gimbal angle trajectories of the 2/4 CMG configuration, assuming the CMG momentum was desaturated.

The Gauss pseudospectral method has the advantage that the cost function can be varied as desired for mission-specific preferences, whether greater emphasis should be placed on minimum time or maximal singularity avoidance. The main disadvantage of the Gauss pseudospectral method is that it is an open-loop approach; the method is trajectory planning for before the maneuver, not during. Also, it is sometimes difficult for the NLP to converge if the initial guess is not proper or if the CMG momentum is

near saturation. Recent studies from Hart et al. [21] suggest the Birkhoff pseudospectral method is more robust to the initial guess than the Gauss pseudospectral method. In addition, further study should hope to find closed-loop steering for the 3/4 pyramid configuration and/or methods to compute optimal trajectories more robustly to the initial guess. With stronger failure-accommodating steering, the 4-CMG configuration will become more and more faithful to implement in reality.

Bibliography

[1] Wie, B., Bailey, D., and Heiberg, C. "Singularity Robust Steering Logic for Redundant Single-Gimbal Control Moment Gyros". *Journal of Guidance, Control, and Dynamics*, 24(5):865–872, 2001. 2, 3, 15, 16, 26, 27, 32, 35

[2] Leve, F., Hamilton, B., and Peck, M. *Spacecraft Momentum Control Systems*. Springer International Publishing, Switzerland, 2015. 2

[3] Kurokawa, H. "Survey of Theory and Steering Laws of Single-Gimbal Control Moment Gyros". *Journal of Guidance, Control, and Dynamics*, 30(5):1331–1340, 2007. 2, 3

[4] Wie, B. *Space Vehicle Dynamics and Control*. American Institute of Aeronautics and Astronautics, Inc., Virginia, US, 2nd edition, 2008. 3

[5] Vadali, S., Oh, H., and Walker, S. "Preferred Gimbal Angles for Single Gimbal Control Moment Gyros". *Journal of Guidance, Control, and Dynamics*, 13(6):1090–1095, 1990. 3, 15

[6] Lee, D. and Bang, H. "Gimbal-Angle Vectors of the Nonredundant CMG Cluster". *International Journal of Aeronautical Space Sciences*, 19:443–458, 2018. 3

[7] Wie, B., Bailey, D., and Heiberg, C. "Rapid Multitarget Acquisition and Pointing Control of Agile Spacecraft". *Journal of Guidance, Control, and Dynamics*, 25(1):96–104, 2002. 3, 26

[8] Fleming, A. and Ross, M. "Singularity-Free Optimal Steering of Control Moment Gyros". *Advances in the Astronautical Sciences*, 123:2681–2700, 2006. 3, 4, 5

[9] Paradiso, J. "Global Steering of Single Gimballed Control Moment Gyroscopes Using a Directed Search". In *Guidance and Control Conference*. AIAA, 1991. 3, 4, 5

[10] Sun, Z. and Ding, S. "SGCMG Non-singularity Steering Based on Adaptive Gauss Pseudospectral Method". In *International Conference on Informative and Cybernetics for Computational Social Systems (ICCSS)*. IEEE, October 2014. 3, 4, 5

[11] Zhang, W., Zhang, Y., Li, W., and Wang, Y. "Path Planning for Rapid Large-Angle Maneuver of Satellites Based on the Gauss Pseudospectral Method". *Mathematical Problems in Engineering*, 2016. Hindawi Publishing Corporation. 3, 4, 5

[12] Meldrum, A., Nonomura, S., Yamada, K., and Shoji, Y. "Attitude Control Using Three Control Moment Gyros". *Japan Society for Aeronautical and Space Sciences*, 16(5):405–411, 2018. 3

[13] Lee, D., Park, C., and Bang, H. "Gimbal Angle Reorientation for Nonredundant Single Gimbal Control Moment Gyros". In *Guidance, Navigation, and Control (GNC) Conference*. AIAA, 2013. 3, 4, 5

[14] Sands, T., Kim, J., and Agrawal, B. "Nonredundant Single-Gimbaled Control Moment Gyroscopes". *Journal of Guidance, Control, and Dynamics*, 35(2):578–587, 2012. 3

[15] Gurrisi, C., et al. "Space Station Control Moment Gyroscope Lessons Learned". In *40th Aerospace Mechanisms Symposium*. NASA Kennedy Space Center, May 2010. 3

[16] SpaceNews Staff. "DigitalGlobe loses WorldView-4 satellite to gyro failure". *SpaceNews.com*, Jan. 7, 2019. 3

[17] Benson, D. *A Gauss Pseudospectral Transcription for Optimal Control*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2005. 4, 9

[18] Benson, D., Huntington, G., Thorvaldsen, T., and Rao, A. "Direct Trajectory Optimization and Costate Estimation via an Orthogonal Collocation Method". *Journal of Guidance, Control, and Dynamics*, 29(6):1435–1440, 2006. 4, 9

[19] Herber, Daniel R. optimal-control-direct-method-examples. *GitHub: https://github.com/danielrherber/optimal-control-direct-method-examples*, 2019. 5

[20] Wie, B. and Lu, J. "Feedback Control Logic for Spacecraft Eigenaxis Rotations Under Slew Rate and Control Constraints". *Journal of Guidance, Control, and Dynamics*, 18(6):1372–1379, 1995. 6

[21] Hart, S., Ayoubi, M., and Naseradinmousavi, P. "Comparative Study of Pseudospectral Methods for Spacecraft Optimal Attitude Maneuvers". *Journal of Spacecraft and Rockets*, 2021. 9, 39

[22] Betts, J. "Survey of Numerical Methods for Trajectory Optimization". *Journal of Guidance, Control, and Dynamics*, 21(2):193–207, 1998. 9

[23] Hakim, V. and Ayoubi, M. "Optimal Steering of Nonredundant Single-Gimbal CMGs using Gauss Pseudospectral Method". In *IEEE Aerospace Conference*. IEEE, 2021.

[24] Markley, F. and Crassidis, J. *Fundamentals of Spacecraft Attitude Determination and Control*. Springer, New York, US, 2014.

[25] Geng, Y., Hou, Z., and Huang, S. "Global Singularity Avoidance Steering Law for Single-Gimbal Control Moment Gyroscopes". *Journal of Guidance, Control, and Dynamics*, 40(2), 2017.

Appendix A

Clarification of Spacecraft Tensor J

In much of the literature on CMGs, details on the precise definition of the spacecraft tensor J are often omitted. Here, we will present a short derivation of equation (1.2) and a precise definition of J.

It is a well-known result of Euler that the total angular momentum $\mathbf{H}$ of a system, with coordinates taken in a reference frame rotating with angular velocity $\boldsymbol{\omega}$, satisfies the equation

$$\dot{\mathbf{H}} + \boldsymbol{\omega} \times \mathbf{H} = \mathbf{T}_{\text{ext}}. \tag{A.1}$$

Thus equation (1.2) relies on the fact that the total angular momentum of the spacecraft-CMG system about the spacecraft's center of mass and in the space frame is $\mathbf{H} = J\boldsymbol{\omega} + \boldsymbol{h}$. Let there be n CMG rotors. Let J_s be the tensor of inertia for only the spacecraft body, not including the CMG rotors. Let $\boldsymbol{h}^{\text{total}}$ be the total angular momentum of the CMG rotors. Then the total angular momentum of the spacecraft-CMG system is

$$\mathbf{H} = J_s\boldsymbol{\omega} + \boldsymbol{h}^{\text{total}}. \tag{A.2}$$

Of course, $\boldsymbol{h}^{\text{total}}$ is the sum of each individual CMG's total angular momentum:

$$\boldsymbol{h}^{\text{total}} = \sum_{i=1}^{n} \boldsymbol{h}_i^{\text{total}}. \tag{A.3}$$

Each CMG's total angular momentum is the angular momentum of its center of mass

plus the angular momentum of itself about its center of mass:

$$\boldsymbol{h}_i^{\text{total}} = \boldsymbol{r}_i \times m_i \boldsymbol{v}_i + J_i \boldsymbol{\omega}_i^{\text{total}}. \tag{A.4}$$

Here, m_i is the mass of the ith CMG rotor, $\boldsymbol{r}_i$ is the position vector of the ith CMG rotor's center of mass relative to the spacecraft's center of mass, and $\boldsymbol{v}_i$ is the velocity of the ith CMG rotor's center of mass relative to the spacecraft's center of mass. J_i is the tensor of inertia of the ith CMG rotor about its center of mass (not about the spacecraft center of mass), and $\boldsymbol{\omega}_i^{\text{total}}$ is the total angular velocity of the ith CMG rotor. Notice:

$$\begin{aligned} \boldsymbol{r}_i \times m_i \boldsymbol{v}_i &= m_i \, \boldsymbol{r}_i \times (\boldsymbol{\omega} \times \boldsymbol{r}_i) \\ &= -m_i \, \boldsymbol{r}_i \times (\boldsymbol{r}_i \times \boldsymbol{\omega}) \\ &= -m_i \, \boldsymbol{r}_{i\times}^{\,2} \, \boldsymbol{\omega}, \end{aligned} \tag{A.5}$$

where $\boldsymbol{r}_{i\times}$ is a 3-by-3 matrix written in terms of its components:

$$\boldsymbol{r}_{i\times} = \begin{bmatrix} 0 & -r_{iz} & r_{iy} \\ r_{iz} & 0 & -r_{ix} \\ -r_{iy} & r_{ix} & 0 \end{bmatrix}. \tag{A.6}$$

Next, note that $\boldsymbol{\omega}_i^{\text{total}}$ is the spacecraft's angular velocity $\boldsymbol{\omega}$ plus the ith CMG rotor's axial angular velocity (relative to the gimbal) $\boldsymbol{\omega}_i^{(r)}$ plus the ith CMG's gimbal velocity (relative to the spacecraft) $\boldsymbol{\omega}_i^{(g)}$:

$$\boldsymbol{\omega}_i^{\text{total}} = \boldsymbol{\omega} + \boldsymbol{\omega}_i^{(r)} + \boldsymbol{\omega}_i^{(g)}. \tag{A.7}$$

Plugging equations (A.3), (A.4), (A.5), and (A.7) back into equation (A.2), we get

$$\begin{aligned}\mathbf{H} &= J_s\boldsymbol{\omega} + \sum_{i=1}^{n}\Big(- m_i\, \boldsymbol{r}_{i\times}^2\, \boldsymbol{\omega} + J_i\big(\boldsymbol{\omega} + \boldsymbol{\omega}_i^{(r)} + \boldsymbol{\omega}_i^{(g)}\big)\Big) \\ &= \left[J_s + \sum_{i=1}^{n}\Big(J_i - m_i\, \boldsymbol{r}_{i\times}^2\Big)\right]\boldsymbol{\omega} + \sum_{i=1}^{n} J_i\big(\boldsymbol{\omega}_i^{(r)} + \boldsymbol{\omega}_i^{(g)}\big).\end{aligned} \tag{A.8}$$

Note that the summation $\sum_{i=1}^{n} J_i\big(\boldsymbol{\omega}_i^{(r)} + \boldsymbol{\omega}_i^{(g)}\big)$ is exactly $\boldsymbol{h}$ as defined in Section 1.1, as long as we have the assumption that each CMG rotor's center of mass is fixed within the spacecraft at all times. We arrive at our final desired result:

$$\mathbf{H} = J\boldsymbol{\omega} + \boldsymbol{h}, \tag{A.9}$$

with

$$J \doteq J_s + \sum_{i=1}^{n}\Big(J_i - m_i\, \boldsymbol{r}_{i\times}^2\Big). \tag{A.10}$$

In Section 1.1, we made the very common assumption that each CMG's gimbal velocity is negligible compared to its axial velocity. That is, we assume $\boldsymbol{\omega}_i^{(g)}$ is negligible compared to $\boldsymbol{\omega}_i^{(r)}$. This is especially reasonable considering the fact that a rotor's moment of inertia is usually largest in the axial direction.

Take notice that the J_i terms are *not* constant—they depend on the CMG rotors' orientations relative to the spacecraft. Thus J is technically not constant. However, it is common to make the approximation that J is constant.

Appendix B

Code for Gauss Pseudospectral Method

Applied to the 3/4 CMG configuration:

%%

```
%--------------------------------------------------------------------------
% CMG3.m
% Solve the 3/4-CMG pyramid configuration using
% the differential Gauss pseudospectral method
% (namely Gauss nodes and Gaussian quadrature)
% Method developed by David Benson, MIT PhD thesis
%--------------------------------------------------------------------------
%
%--------------------------------------------------------------------------
% Contributor: Victor Hakim
% Underlying source code by:
% Daniel R. Herber, Graduate Student, University
% of Illinois at Urbana-Champaign, website:
% https://github.com/danielrherber/optimal-control-direct-method-examples
%--------------------------------------------------------------------------

close all
clear all

% Run Wie simulation, for comparison
WieSimulation
close all

% Problem parameters
p.ns = 10; p.nu = 3; % number of states and controls
p.t0 = 0;            % time horizon (final time to be determined)
p.j = diag([21400,20100,5000]); % spacecraft tensor of inertia, in SI units
p.jInv = inv(p.j);
p.beta = 53.13; % skew angle, in degrees
p.h = 1000; % each CMG momentum, in SI units (constant)
slewMax = 10; % max slew rate, in deg/s
p.slewMax = slewMax*pi/180; % convert to rad/s
p.deltaMax = 2; % max gimbal rate, in rad/s
p.q0 = [0; 0; 0; 1]; % quaternion initial condition
p.qf = [0.4; 0; 0; sqrt(0.84)]; % quaternion final condition (Case A)
% p.qf = (0.5/sqrt(3))*[1; 1; 1; 3]; % quaternion final condition (Case B)
p.w0 = [0; 0; 0]; % angular velocity initial condition
p.wf = [0; 0; 0]; % angular velocity final condition
delta0 = [60; 180; -60]; % initial gimbal angles, in degrees (Case 1)
% delta0 = [90; 0; -90]; % singularity (Case 2)
p.delta0 = delta0*pi/180; % convert to radians
p.tfGuess = 9; % guess of final time, in seconds (Case 1: 9, Case 2: 11)
p.tfUpper = 12; % upper bound on maneuver time (Case 1: 12, Case 2: 15)

% Direct transcription parameters
% pre-computed values for tau,w,D available for N = 10,20,50,100
% p.nt = 10; % number of node points, including -1
p.nt = 20; % number of node points, including -1
% p.nt = 50; % number of node points, including -1
p.tau = double(Gauss_nodes(p.nt-1)); % scaled time horizon
p.D =  double(Gauss_Dmatrix(p.tau)); % differential approximation matrix
p.W = double(Gauss_weights(p.tau)); % for Gaussian quadrature

% Discretized variable indices in x = [q1;...w1;...d1;...;u1...];
n = p.nt;
p.q1i = 1:n; p.q2i = n+1:2*n;
p.q3i = 2*n+1:3*n; p.q4i = 3*n+1:4*n; m = 4*n;
p.w1i = m+1:m+n; p.w2i = m+n+1:m+2*n;
p.w3i = m+2*n+1:m+3*n; m = m+3*n;
p.d1i = m+1:m+n; p.d2i = m+n+1:m+2*n;
p.d3i = m+2*n+1:m+3*n; m = m+3*n;
p.u1i = m+1:m+n-1; p.u2i = m+n:m+2*n-2;
p.u3i = m+2*n-1:m+3*n-3;
```

```
p.tfi = m+3*n-2;
len = m+3*n-2; % length of x

% Initial guess
x0 = initialGuess(Wie,p,len);

% Optimization algorithm options
options = optimoptions(@fmincon,'Display','iter','MaxFunEvals',1e6, ...
    'Algorithm','sqp','MaxIterations',1000);

% Solve the problem
% x = fmincon(@(x) objective(x,p),x0,[],[],[],[],[],[],@(x) constraints(x,p,2),options);
% x0 = x;
% x = fmincon(@(x) objective(x,p),x0,[],[],[],[],[],[],@(x) constraints(x,p,3),options);
% x0 = x;
x = fmincon(@(x) objective(x,p),x0,[],[],[],[],[],[],@(x) constraints(x,p,1),options);

% temporary
% x = x0;

% Extract the optimal solution
q1 = x(p.q1i); q2 = x(p.q2i); q3 = x(p.q3i); q4 = x(p.q4i);
w1 = x(p.w1i); w2 = x(p.w2i); w3 = x(p.w3i);
d1 = x(p.d1i); d2 = x(p.d2i); d3 = x(p.d3i);
u1 = x(p.u1i); u2 = x(p.u2i); u3 = x(p.u3i);
q = [q1, q2, q3, q4];
w = [w1, w2, w3];
delta = [d1, d2, d3];
u = [u1, u2, u3];
p.tf = x(p.tfi); % extract final time
p.t = (p.tau*(p.tf-p.t0) + (p.tf+p.t0))/2; % unscale time horizon

% Calculate (1) determinant of Jacobian over time and
% (2) system angular momentum magnitude over time
Det = zeros(length(d1),1);
HSystem = zeros(length(d1),1);
for it = 1:length(d1)
    Det(it) = det(Jacobian3(p.h, p.beta, [d1(it);d2(it);d3(it)]*180/pi));
    HSystem(it) = norm(p.j*w(it,:)' +  ...
        angVector3(p.h, p.beta, [d1(it);d2(it);d3(it)]*180/pi));
end

% Obtain the costate
% Lam = zeros(p.nt + 1, 1);
% Lam(1) = lambda.eqnonlin(1);
% Lam(end) = -lambda.eqnonlin(2);
% Lam(2:end-1) = lambda.eqnonlin(3:end)./p.w + Lam(end);
% Lam = -Lam;

% Plots
% Plots_Gauss_CMG3(q,w,delta,u,Det,p,'Pseudospectral')
Plots2_Gauss_CMG3(q,w,delta,u,Det,HSystem,p,Wie,'Pseudospectral') % for paper

% Display time of completion
datestr(now)

% Initial guess
function x0 = initialGuess(Wie,p,len)
    x0 = zeros(len,1);
    indices = round(0.5*(p.tau+1)*p.tfGuess/Wie.dt);
    indices = [1; indices];
    x0(p.q1i) = Wie.qArray(1,indices)';
    x0(p.q2i) = Wie.qArray(2,indices)';
    x0(p.q3i) = Wie.qArray(3,indices)';
    x0(p.q4i) = Wie.qArray(4,indices)';
    x0(p.w1i) = Wie.wArray(1,indices)';
```

```
    x0(p.w2i) = Wie.wArray(2,indices)';
    x0(p.w3i) = Wie.wArray(3,indices)';
    x0(p.tfi) = p.tfGuess;
    x0(p.d1i) = p.delta0(1);
    x0(p.d2i) = p.delta0(2);
    x0(p.d3i) = p.delta0(3);

    % temporary (A2)
%     x0(p.w1i) = x0(p.w1i)*180/pi;
%     x0(p.w2i) = x0(p.w2i)*180/pi;
%     x0(p.w3i) = x0(p.w3i)*180/pi;
end

% Objective function
function J = objective(x,p)
    % Extract
    tf = x(p.tfi); % extract final time
    d1 = x(p.d1i); d2 = x(p.d2i); d3 = x(p.d3i); % extract gimbal angles

    % Calculate determinant of Jacobian over time
    Det = zeros(length(d1),1);
    for it = 1:length(d1)
        Det(it) = det(Jacobian3(p.h, p.beta, [d1(it);d2(it);d3(it)]*180/pi));
    end
    Det = Det(2:end);

    % Weights
    Weight1 = 0; % for final time
%     Weight2 = 0; % for maximal integrated det(A)
    Weight2 = 1e-14;

    L = -Weight2*abs(Det); % integrand
    J = Weight1*tf + ((tf-p.t0)/2)*dot(p.W,L); % cost function
end

% Constraint function
function [c,ceq] = constraints(x,p,choice)
    % Extract
    q1 = x(p.q1i); q2 = x(p.q2i); q3 = x(p.q3i); q4 = x(p.q4i);
    w1 = x(p.w1i); w2 = x(p.w2i); w3 = x(p.w3i);
    d1 = x(p.d1i); d2 = x(p.d2i); d3 = x(p.d3i);
    u1 = x(p.u1i); u2 = x(p.u2i); u3 = x(p.u3i);
    tf = x(p.tfi); % extract final time

    qArray = [q1'; q2'; q3'; q4'];
    wArray = [w1'; w2'; w3'];
    deltaArray = [d1'; d2'; d3'];
    uArray = [u1'; u2'; u3'];

    qvDotArray = zeros(3, p.nt-1);
    q4DotArray = zeros(1, p.nt-1);
    wDotArray = zeros(3, p.nt-1);

    % Equations of motion
    for it = 2:p.nt
        qvDotArray(:,it-1) = -0.5*cross(wArray(:,it), qArray(1:3,it)) ...
            + 0.5*qArray(4,it)*wArray(:,it);
        q4DotArray(it-1) = -0.5*dot(wArray(:,it), qArray(1:3,it));

        hv = angVector3(p.h, p.beta, deltaArray(:,it)*180/pi);
        A = Jacobian3(p.h, p.beta, deltaArray(:,it)*180/pi);

        wDotArray(:,it-1) = -p.jInv*(cross(wArray(:,it), p.j*wArray(:,it)+hv) ...
            + A*uArray(:,it-1));
    end
```

```
    % Create matrices: (p.nt x p.ns) for Y, one fewer row for F.
    % For Gauss pseudospectral, k=0 is needed here:
    Y = [q1,q2,q3,q4,w1,w2,w3,d1,d2,d3];
    % For Gauss PS, no k=0 here:
    F = ((tf-p.t0)/2)*[qvDotArray', q4DotArray', wDotArray', uArray'];

    % Initial state conditions
    ceq1 = qArray(:,1) - p.q0;
    ceq2 = wArray(:,1) - p.w0;
    ceq3 = deltaArray(:,1) - p.delta0;

    % Final state conditions
    % necessary for *Gauss* pseudospectral method
    ceq4 = qArray(:,1) + dot([p.W,p.W,p.W,p.W],F(:,1:4))' - p.qf;
    ceq5 = wArray(:,1) + dot([p.W,p.W,p.W],F(:,5:7))' - p.wf;

    ceq6 = p.D*Y - F; % differential equation approximation

    ceq7 = vecnorm(qArray)' - 1; % quaternion condition

    % Calculate system angular momentum magnitude
    HSystem = zeros(length(d1),1);
    for it = 1:length(d1)
        HSystem(it) = norm(p.j*wArray(:,it) +  ...
            angVector3(p.h, p.beta, [d1(it);d2(it);d3(it)]*180/pi));
    end

    % Path constraints
    c1 = abs(u1) - p.deltaMax;
    c2 = abs(u2) - p.deltaMax;
    c3 = abs(u3) - p.deltaMax;
    c4 = vecnorm(wArray)' - p.slewMax;
    c5 = -tf;
    c6 = tf - p.tfUpper;

    % Keep angular momentum constant
    c7 = abs(HSystem - HSystem(1)) - 0.001*HSystem(1);

    % Combine constraints
    if choice == 0
        c = [c1;c2;c3;c4;c5;c6];
        ceq = [ceq1;ceq2;ceq3;ceq4;ceq5;ceq6(:);ceq7];
    elseif choice == 1
        c = [c1;c2;c3;c4;c5;c6;c7];
        ceq = [ceq1;ceq2;ceq3;ceq4;ceq5;ceq6(:)];
    elseif choice == 2
        c = [c1;c2;c3;c4;c5;c6];
        ceq = [ceq1;ceq2;ceq3;ceq4;ceq6(:)];
    elseif choice == 3
        c = [c1;c2;c3;c4;c5;c6];
        ceq = [ceq1;ceq2;ceq3;ceq4;ceq6(:);ceq7];
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Applied to the 2/4 adjacent CMG configuration:

%%%

```
%--------------------------------------------------------------------------
% CMG12.m
% Solve the 2/4-CMG pyramid adjacent configuration using
% the differential Gauss pseudospectral method
% (namely Gauss nodes and Gaussian quadrature)
% Method developed by David Benson, MIT PhD thesis
%--------------------------------------------------------------------------
%
%--------------------------------------------------------------------------
% Contributor: Victor Hakim
% Underlying source code by:
% Daniel R. Herber, Graduate Student, University
% of Illinois at Urbana-Champaign, website:
% https://github.com/danielrherber/optimal-control-direct-method-examples
%--------------------------------------------------------------------------

close all
clear all

% Run Wie simulation, for comparison
WieSimulation
close all

% Problem parameters
p.ns = 9; p.nu = 2; % number of states and controls
p.t0 = 0;             % time horizon (final time to be determined)
p.j = diag([21400,20100,5000]); % spacecraft tensor of inertia, in SI units
p.jInv = inv(p.j);
p.beta = 53.13; % skew angle, in degrees
p.h = 1000; % each CMG momentum, in SI units (constant)
slewMax = 10; % max slew rate, in deg/s
p.slewMax = slewMax*pi/180; % convert to rad/s
p.deltaMax = 2; % max gimbal rate, in rad/s
p.q0 = [0; 0; 0; 1]; % quaternion initial condition
p.qf = [0.4; 0; 0; sqrt(0.84)]; % quaternion final condition (Case A)
% p.qf = (0.5/sqrt(3))*[1; 1; 1; 3]; % quaternion final condition (Case B)
p.w0 = [0; 0; 0]; % angular velocity initial condition
p.wf = [0; 0; 0]; % angular velocity final condition
delta0 = [90; -90]; % initial gimbal angles, in degrees
p.delta0 = delta0*pi/180; % convert to radians
p.tfGuess = 35; % guess of final time, in seconds

% Direct transcription parameters
% pre-computed values for tau,w,D available for N = 10,20,50,100
% p.nt = 10; % number of node points, including -1
p.nt = 20; % number of node points, including -1
% p.nt = 50; % number of node points, including -1
p.tau = double(Gauss_nodes(p.nt-1)); % scaled time horizon
p.D =  double(Gauss_Dmatrix(p.tau)); % differential approximation matrix
p.W = double(Gauss_weights(p.tau)); % for Gaussian quadrature

% Discretized variable indices in x = [q1;...w1;...d1;...;u1...];
n = p.nt;
p.q1i = 1:n; p.q2i = n+1:2*n;
p.q3i = 2*n+1:3*n; p.q4i = 3*n+1:4*n; m = 4*n;
p.w1i = m+1:m+n; p.w2i = m+n+1:m+2*n;
p.w3i = m+2*n+1:m+3*n; m = m+3*n;
p.d1i = m+1:m+n; p.d2i = m+n+1:m+2*n; m = m+2*n;
p.u1i = m+1:m+n-1; p.u2i = m+n:m+2*n-2;
p.tfi = m+2*n-1;
len = m+2*n-1; % length of x

% Initial guess
```

```
x0 = initialGuess(Wie,p,len);

% Optimization algorithm options
options = optimoptions(@fmincon,'Display','iter','MaxFunEvals',1e6, ...
    'Algorithm','sqp','MaxIterations',1000);

% Solve the problem
% x = fmincon(@(x) objective(x,p),x0,[],[],[],[],[],[],@(x) constraints(x,p,2),options);
% x0 = x;
% x = fmincon(@(x) objective(x,p),x0,[],[],[],[],[],[],@(x) constraints(x,p,3),options);
% x0 = x;
x = fmincon(@(x) objective(x,p),x0,[],[],[],[],[],[],@(x) constraints(x,p,1),options);

% temporary
% x = x0;

% Extract the optimal solution
q1 = x(p.q1i); q2 = x(p.q2i); q3 = x(p.q3i); q4 = x(p.q4i);
w1 = x(p.w1i); w2 = x(p.w2i); w3 = x(p.w3i);
d1 = x(p.d1i); d2 = x(p.d2i);
u1 = x(p.u1i); u2 = x(p.u2i);
q = [q1, q2, q3, q4];
w = [w1, w2, w3];
delta = [d1, d2];
u = [u1, u2];
p.tf = x(p.tfi); % extract final time
p.t = (p.tau*(p.tf-p.t0) + (p.tf+p.t0))/2; % unscale time horizon

% Calculate (1) determinant of Jacobian over time and
% (2) system angular momentum magnitude over time
% Det = zeros(length(d1),1);
HSystem = zeros(length(d1),1);
for it = 1:length(d1)
%     Det(it) = det(Jacobian3(p.h, p.beta, [d1(it);d2(it);d3(it)]*180/pi));
    HSystem(it) = norm(p.j*w(it,:)' +  ...
        angVector2adj(p.h, p.beta, [d1(it);d2(it)]*180/pi));
end

% Obtain the costate
% Lam = zeros(p.nt + 1, 1);
% Lam(1) = lambda.eqnonlin(1);
% Lam(end) = -lambda.eqnonlin(2);
% Lam(2:end-1) = lambda.eqnonlin(3:end)./p.w + Lam(end);
% Lam = -Lam;

% Plots
% Plots_Gauss_CMG3(q,w,delta,u,Det,p,'Pseudospectral')
Plots2_Gauss_CMG12(q,w,delta,u,HSystem,p,Wie,'Pseudospectral') % for paper

% Display time of completion
datestr(now)

% Initial guess
function x0 = initialGuess(Wie,p,len)
    x0 = zeros(len,1);
%     indices = round(0.5*(p.tau+1)*p.tfGuess/Wie.dt);
    indices = round(0.5*(p.tau+1)*10/Wie.dt);
    indices = [1; indices];
    x0(p.q1i) = Wie.qArray(1,indices)';
    x0(p.q2i) = Wie.qArray(2,indices)';
    x0(p.q3i) = Wie.qArray(3,indices)';
    x0(p.q4i) = Wie.qArray(4,indices)';
    x0(p.w1i) = Wie.wArray(1,indices)';
    x0(p.w2i) = Wie.wArray(2,indices)';
    x0(p.w3i) = Wie.wArray(3,indices)';
    x0(p.tfi) = p.tfGuess;
```

```
    x0(p.d1i) = p.delta0(1);
    x0(p.d2i) = p.delta0(2);

    % temporary (A2)
%     x0(p.w1i) = x0(p.w1i)*180/pi;
%     x0(p.w2i) = x0(p.w2i)*180/pi;
%     x0(p.w3i) = x0(p.w3i)*180/pi;
end

% Objective function
function J = objective(x,p)
    % Extract
    tf = x(p.tfi); % extract final time
%     d1 = x(p.d1i); d2 = x(p.d2i); d3 = x(p.d3i); % extract gimbal angles

    % Calculate determinant of Jacobian over time
%     Det = zeros(length(d1),1);
%     for it = 1:length(d1)
%         Det(it) = det(Jacobian3(p.h, p.beta, [d1(it);d2(it);d3(it)]*180/pi));
%     end
%     Det = Det(2:end);

    % Weights
%     Weight1 = 0; % for final time
%     Weight2 = 0; % for maximal integrated det(A)
%     Weight2 = 1e-14;

%     L = -Weight2*abs(Det); % integrand
    J = tf; % cost function
end

% Constraint function
function [c,ceq] = constraints(x,p,choice)
    % Extract
    q1 = x(p.q1i); q2 = x(p.q2i); q3 = x(p.q3i); q4 = x(p.q4i);
    w1 = x(p.w1i); w2 = x(p.w2i); w3 = x(p.w3i);
    d1 = x(p.d1i); d2 = x(p.d2i);
    u1 = x(p.u1i); u2 = x(p.u2i);
    tf = x(p.tfi); % extract final time

    qArray = [q1'; q2'; q3'; q4'];
    wArray = [w1'; w2'; w3'];
    deltaArray = [d1'; d2'];
    uArray = [u1'; u2'];

    qvDotArray = zeros(3, p.nt-1);
    q4DotArray = zeros(1, p.nt-1);
    wDotArray = zeros(3, p.nt-1);

    % Equations of motion
    for it = 2:p.nt
        qvDotArray(:,it-1) = -0.5*cross(wArray(:,it), qArray(1:3,it)) ...
            + 0.5*qArray(4,it)*wArray(:,it);
        q4DotArray(it-1) = -0.5*dot(wArray(:,it), qArray(1:3,it));

        hv = angVector2adj(p.h, p.beta, deltaArray(:,it)*180/pi);
        A = Jacobian2adj(p.h, p.beta, deltaArray(:,it)*180/pi);

        wDotArray(:,it-1) = -p.jInv*(cross(wArray(:,it), p.j*wArray(:,it)+hv) ...
            + A*uArray(:,it-1));
    end

    % Create matrices: (p.nt x p.ns) for Y, one fewer row for F.
    % For Gauss pseudospectral, k=0 is needed here:
    Y = [q1,q2,q3,q4,w1,w2,w3,d1,d2];
    % For Gauss PS, no k=0 here:
```

```
    F = ((tf-p.t0)/2)*[qvDotArray', q4DotArray', wDotArray', uArray'];

    % Initial state conditions
    ceq1 = qArray(:,1) - p.q0;
    ceq2 = wArray(:,1) - p.w0;
    ceq3 = deltaArray(:,1) - p.delta0;

    % Final state conditions
    % necessary for *Gauss* pseudospectral method
    ceq4 = qArray(:,1) + dot([p.W,p.W,p.W,p.W],F(:,1:4))' - p.qf;
    ceq5 = wArray(:,1) + dot([p.W,p.W,p.W],F(:,5:7))' - p.wf;

    ceq6 = p.D*Y - F; % differential equation approximation

    ceq7 = vecnorm(qArray)' - 1; % quaternion condition

    % Calculate system angular momentum magnitude
    HSystem = zeros(length(d1),1);
    for it = 1:length(d1)
        HSystem(it) = norm(p.j*wArray(:,it) +  ...
            angVector2adj(p.h, p.beta, [d1(it);d2(it)]*180/pi));
    end

    % Path constraints
    c1 = abs(u1) - p.deltaMax;
    c2 = abs(u2) - p.deltaMax;
    c3 = -1;
    c4 = vecnorm(wArray)' - p.slewMax;
    c5 = -tf;
%     c6 = tf - p.tfUpper;
    c6 = -1;

    % Keep angular momentum constant
    c7 = abs(HSystem - HSystem(1)) - 0.001*HSystem(1);

    % Combine constraints
    if choice == 0
        c = [c1;c2;c3;c4;c5;c6];
        ceq = [ceq1;ceq2;ceq3;ceq4;ceq5;ceq6(:);ceq7];
    elseif choice == 1
        c = [c1;c2;c3;c4;c5;c6;c7];
        ceq = [ceq1;ceq2;ceq3;ceq4;ceq5;ceq6(:)];
    elseif choice == 2
        c = [c1;c2;c3;c4;c5;c6];
        ceq = [ceq1;ceq2;ceq3;ceq4;ceq6(:)];
    elseif choice == 3
        c = [c1;c2;c3;c4;c5;c6];
        ceq = [ceq1;ceq2;ceq3;ceq4;ceq6(:);ceq7];
    end
end
```

%%

Applied to the 2/4 opposite CMG configuration:

%%

```
%-------------------------------------------------------------------------
% CMG13.m
% Solve the 2/4-CMG pyramid opposite configuration using
% the differential Gauss pseudospectral method
% (namely Gauss nodes and Gaussian quadrature)
% Method developed by David Benson, MIT PhD thesis
%-------------------------------------------------------------------------
%
%-------------------------------------------------------------------------
% Contributor: Victor Hakim
% Underlying source code by:
% Daniel R. Herber, Graduate Student, University
% of Illinois at Urbana-Champaign, website:
% https://github.com/danielrherber/optimal-control-direct-method-examples
%-------------------------------------------------------------------------

close all
clear all

% Run Wie simulation, for comparison
WieSimulation
close all

% Problem parameters
p.ns = 9; p.nu = 2; % number of states and controls
p.t0 = 0;            % time horizon (final time to be determined)
p.j = diag([21400,20100,5000]); % spacecraft tensor of inertia, in SI units
p.jInv = inv(p.j);
p.beta = 53.13; % skew angle, in degrees
p.h = 1000; % each CMG momentum, in SI units (constant)
slewMax = 10; % max slew rate, in deg/s
p.slewMax = slewMax*pi/180; % convert to rad/s
p.deltaMax = 2; % max gimbal rate, in rad/s
p.q0 = [0; 0; 0; 1]; % quaternion initial condition
p.qf = [0.4; 0; 0; sqrt(0.84)]; % quaternion final condition (Case A)
% p.qf = (0.5/sqrt(3))*[1; 1; 1; 3]; % quaternion final condition (Case B)
p.w0 = [0; 0; 0]; % angular velocity initial condition
p.wf = [0; 0; 0]; % angular velocity final condition
delta0 = [10; -10]; % initial gimbal angles, in degrees
p.delta0 = delta0*pi/180; % convert to radians
p.tfGuess = 35; % guess of final time, in seconds

% Direct transcription parameters
% pre-computed values for tau,w,D available for N = 10,20,50,100
% p.nt = 10; % number of node points, including -1
p.nt = 20; % number of node points, including -1
% p.nt = 50; % number of node points, including -1
p.tau = double(Gauss_nodes(p.nt-1)); % scaled time horizon
p.D =  double(Gauss_Dmatrix(p.tau)); % differential approximation matrix
p.W = double(Gauss_weights(p.tau)); % for Gaussian quadrature

% Discretized variable indices in x = [q1;...w1;...d1;...;u1...];
n = p.nt;
p.q1i = 1:n; p.q2i = n+1:2*n;
p.q3i = 2*n+1:3*n; p.q4i = 3*n+1:4*n; m = 4*n;
p.w1i = m+1:m+n; p.w2i = m+n+1:m+2*n;
p.w3i = m+2*n+1:m+3*n; m = m+3*n;
p.d1i = m+1:m+n; p.d2i = m+n+1:m+2*n; m = m+2*n;
p.u1i = m+1:m+n-1; p.u2i = m+n:m+2*n-2;
p.tfi = m+2*n-1;
len = m+2*n-1; % length of x

% Initial guess
```

```
x0 = initialGuess(Wie,p,len);

% Optimization algorithm options
options = optimoptions(@fmincon,'Display','iter','MaxFunEvals',1e6, ...
    'Algorithm','sqp','MaxIterations',1000);

% Solve the problem
% x = fmincon(@(x) objective(x,p),x0,[],[],[],[],[],[],@(x) constraints(x,p,2),options);
% x0 = x;
% x = fmincon(@(x) objective(x,p),x0,[],[],[],[],[],[],@(x) constraints(x,p,3),options);
% x0 = x;
x = fmincon(@(x) objective(x,p),x0,[],[],[],[],[],[],@(x) constraints(x,p,1),options);

% temporary
% x = x0;

% Extract the optimal solution
q1 = x(p.q1i); q2 = x(p.q2i); q3 = x(p.q3i); q4 = x(p.q4i);
w1 = x(p.w1i); w2 = x(p.w2i); w3 = x(p.w3i);
d1 = x(p.d1i); d2 = x(p.d2i);
u1 = x(p.u1i); u2 = x(p.u2i);
q = [q1, q2, q3, q4];
w = [w1, w2, w3];
delta = [d1, d2];
u = [u1, u2];
p.tf = x(p.tfi); % extract final time
p.t = (p.tau*(p.tf-p.t0) + (p.tf+p.t0))/2; % unscale time horizon

% Calculate (1) determinant of Jacobian over time and
% (2) system angular momentum magnitude over time
% Det = zeros(length(d1),1);
HSystem = zeros(length(d1),1);
for it = 1:length(d1)
%     Det(it) = det(Jacobian3(p.h, p.beta, [d1(it);d2(it);d3(it)]*180/pi));
    HSystem(it) = norm(p.j*w(it,:)' +  ...
        angVector2opp(p.h, p.beta, [d1(it);d2(it)]*180/pi));
end

% Obtain the costate
% Lam = zeros(p.nt + 1, 1);
% Lam(1) = lambda.eqnonlin(1);
% Lam(end) = -lambda.eqnonlin(2);
% Lam(2:end-1) = lambda.eqnonlin(3:end)./p.w + Lam(end);
% Lam = -Lam;

% Plots
% Plots_Gauss_CMG3(q,w,delta,u,Det,p,'Pseudospectral')
Plots2_Gauss_CMG13(q,w,delta,u,HSystem,p,Wie,'Pseudospectral') % for paper

% Display time of completion
datestr(now)

% Initial guess
function x0 = initialGuess(Wie,p,len)
    x0 = zeros(len,1);
%     indices = round(0.5*(p.tau+1)*p.tfGuess/Wie.dt);
    indices = round(0.5*(p.tau+1)*10/Wie.dt);
    indices = [1; indices];
    x0(p.q1i) = Wie.qArray(1,indices)';
    x0(p.q2i) = Wie.qArray(2,indices)';
    x0(p.q3i) = Wie.qArray(3,indices)';
    x0(p.q4i) = Wie.qArray(4,indices)';
    x0(p.w1i) = Wie.wArray(1,indices)';
    x0(p.w2i) = Wie.wArray(2,indices)';
    x0(p.w3i) = Wie.wArray(3,indices)';
    x0(p.tfi) = p.tfGuess;
```

```
    x0(p.d1i) = p.delta0(1);
    x0(p.d2i) = p.delta0(2);

    % temporary (A2)
%     x0(p.w1i) = x0(p.w1i)*180/pi;
%     x0(p.w2i) = x0(p.w2i)*180/pi;
%     x0(p.w3i) = x0(p.w3i)*180/pi;
end

% Objective function
function J = objective(x,p)
    % Extract
    tf = x(p.tfi); % extract final time
%     d1 = x(p.d1i); d2 = x(p.d2i); d3 = x(p.d3i); % extract gimbal angles

    % Calculate determinant of Jacobian over time
%     Det = zeros(length(d1),1);
%     for it = 1:length(d1)
%         Det(it) = det(Jacobian3(p.h, p.beta, [d1(it);d2(it);d3(it)]*180/pi));
%     end
%     Det = Det(2:end);

    % Weights
%     Weight1 = 0; % for final time
%     Weight2 = 0; % for maximal integrated det(A)
%     Weight2 = 1e-14;

%     L = -Weight2*abs(Det); % integrand
    J = tf; % cost function
end

% Constraint function
function [c,ceq] = constraints(x,p,choice)
    % Extract
    q1 = x(p.q1i); q2 = x(p.q2i); q3 = x(p.q3i); q4 = x(p.q4i);
    w1 = x(p.w1i); w2 = x(p.w2i); w3 = x(p.w3i);
    d1 = x(p.d1i); d2 = x(p.d2i);
    u1 = x(p.u1i); u2 = x(p.u2i);
    tf = x(p.tfi); % extract final time

    qArray = [q1'; q2'; q3'; q4'];
    wArray = [w1'; w2'; w3'];
    deltaArray = [d1'; d2'];
    uArray = [u1'; u2'];

    qvDotArray = zeros(3, p.nt-1);
    q4DotArray = zeros(1, p.nt-1);
    wDotArray = zeros(3, p.nt-1);

    % Equations of motion
    for it = 2:p.nt
        qvDotArray(:,it-1) = -0.5*cross(wArray(:,it), qArray(1:3,it)) ...
            + 0.5*qArray(4,it)*wArray(:,it);
        q4DotArray(it-1) = -0.5*dot(wArray(:,it), qArray(1:3,it));

        hv = angVector2opp(p.h, p.beta, deltaArray(:,it)*180/pi);
        A = Jacobian2opp(p.h, p.beta, deltaArray(:,it)*180/pi);

        wDotArray(:,it-1) = -p.jInv*(cross(wArray(:,it), p.j*wArray(:,it)+hv) ...
            + A*uArray(:,it-1));
    end

    % Create matrices: (p.nt x p.ns) for Y, one fewer row for F.
    % For Gauss pseudospectral, k=0 is needed here:
    Y = [q1,q2,q3,q4,w1,w2,w3,d1,d2];
    % For Gauss PS, no k=0 here:
```

```
    F = ((tf-p.t0)/2)*[qvDotArray', q4DotArray', wDotArray', uArray'];

    % Initial state conditions
    ceq1 = qArray(:,1) - p.q0;
    ceq2 = wArray(:,1) - p.w0;
    ceq3 = deltaArray(:,1) - p.delta0;

    % Final state conditions
    % necessary for *Gauss* pseudospectral method
    ceq4 = qArray(:,1) + dot([p.W,p.W,p.W,p.W],F(:,1:4))' - p.qf;
    ceq5 = wArray(:,1) + dot([p.W,p.W,p.W],F(:,5:7))' - p.wf;

    ceq6 = p.D*Y - F; % differential equation approximation

    ceq7 = vecnorm(qArray)' - 1; % quaternion condition

    % Calculate system angular momentum magnitude
    HSystem = zeros(length(d1),1);
    for it = 1:length(d1)
        HSystem(it) = norm(p.j*wArray(:,it) +  ...
            angVector2opp(p.h, p.beta, [d1(it);d2(it)]*180/pi));
    end

    % Path constraints
    c1 = abs(u1) - p.deltaMax;
    c2 = abs(u2) - p.deltaMax;
    c3 = -1;
    c4 = vecnorm(wArray)' - p.slewMax;
    c5 = -tf;
%     c6 = tf - p.tfUpper;
    c6 = -1;

    % Keep angular momentum constant
    c7 = abs(HSystem - HSystem(1)) - 0.001*HSystem(1);

    % Combine constraints
    if choice == 0
        c = [c1;c2;c3;c4;c5;c6];
        ceq = [ceq1;ceq2;ceq3;ceq4;ceq5;ceq6(:);ceq7];
    elseif choice == 1
        c = [c1;c2;c3;c4;c5;c6;c7];
        ceq = [ceq1;ceq2;ceq3;ceq4;ceq5;ceq6(:)];
    elseif choice == 2
        c = [c1;c2;c3;c4;c5;c6];
        ceq = [ceq1;ceq2;ceq3;ceq4;ceq6(:)];
    elseif choice == 3
        c = [c1;c2;c3;c4;c5;c6];
        ceq = [ceq1;ceq2;ceq3;ceq4;ceq6(:);ceq7];
    end
end
```

%%

Appendix C

Code for Generalized Singularity Robust Simulation

```
% This recreates the simulation of Wie et al. (2001)
% "Singularity Robust Steering Logic for Redundant Single-Gimbal Control
% Moment Gyros"
% Journal Guidance, Control, and Dynamics. Vol. 24, No. 5.

% Setup parameters
beta = 53.13; % skew angle, in degrees
h = 1000; % each CMG momentum, in SI units (constant)
deltaMax = 2; % max gimbal rate, in rad/s
slewMax = 10; % max slew rate, in deg/s
slewMax = slewMax*pi/180; % convert to rad/s
J = diag([21400,20100,5000]); % spacecraft tensor of inertia, in SI units
Jinv = inv(J);
totalTime = 12; % total time of simulation, in seconds
% (A1, B1: 12, A2, B2: 15, adjAB: 50, oppAB: 50)
dt = 1/10000; % interval of time, in seconds

% Control parameters
omegaN = 3; % in rad/s
zeta = 0.9;
T = 1e1; % time constant of integral control, in seconds (should be 10)
k = omegaN^2 + 2*zeta*omegaN/T;
c = 2*zeta*omegaN + 1/T;

% Initial/final conditions
% q: spacecraft quaternion, w: spacecraft angular velocity (body frame),
% delta: gimbal angles, deltaDot: gimbal angle rates
q0 = [0; 0; 0; 1];
qf = [0.4; 0; 0; sqrt(0.84)]; % Case A
% qf = (0.5/sqrt(3))*[1; 1; 1; 3]; % Case B
delta0 = [0; 0; 0; 0]; % Case 1
% delta0 = [90; 0; -90; 0]; % Case 2 (elliptic singularity)
delta0 = delta0*pi/180; % convert to rad
w0 = [0; 0; 0];
deltaDot1D = [0; 0; 0; 0]; % first derivative of deltaDot

% Quaternion error matrix
r1 = [qf(4), qf(3), -qf(2), -qf(1)];
r2 = [-qf(3), qf(4), qf(1), -qf(2)];
r3 = [qf(2), -qf(1), qf(4), -qf(3)];
r4 = [qf(1), qf(2), qf(3), qf(4)];
QEmatrix = [r1; r2; r3; r4];

% Quantities over time
tArray = 0:dt:totalTime;
N = length(tArray); % number of columns for each quantity
qArray = zeros(4,N);
wArray = zeros(3,N);
deltaArray = zeros(4,N);
deltaDotArray = zeros(4,N);

% Initialize
qArray(:,1) = q0;
wArray(:,1) = w0;
deltaArray(:,1) = delta0;
intE = [0; 0; 0]; % integral of error quaternion vector
counter = 0; % for one-time reset of integral


% Calculate maneuver simulation
for it = 1:N-1

    t = tArray(it);
    q = qArray(:,it);
    w = wArray(:,it);
    delta = deltaArray(:,it);
```

```
    deltaDot = deltaDotArray(:,it);
    qv = q(1:3);
    A = Jacobian4(h, beta, delta*180/pi);
    hv = angVector4(h, beta, delta*180/pi);

    % Calculate error quaternion vector
    qErr = QEmatrix*q;
    err = qErr(1:3);

    % Calculate saturated torque (tau) and CMG torque (u)
    limitValues = Limiter(A, err, J, slewMax, deltaMax, k, c);
    errTerm = err + intE/T;
    satErr = [0; 0; 0];
    for it2 = 1:3
        satErr(it2) = sat(errTerm(it2), -limitValues(it2), limitValues(it2));
    end
    tau = -J*(2*k*satErr + c*w);
    u = -tau - cross(w, hv);

    % Calculate generalized robust pseudoinverse (Asharp)
    A = A/h; % to normalize A
    lam = 0.01*exp(-10*det(A*A'));
    eps1 = 0.01*sin(0.5*pi*t);
    eps2 = 0.01*sin(0.5*pi*t + pi/2);
    eps3 = 0.01*sin(0.5*pi*t + pi);
    r1 = [1, eps3, eps2];
    r2 = [eps3, 1, eps1];
    r3 = [eps2, eps1, 1];
    E = [r1; r2; r3];
    Asharp = A'*inv(A*A' + lam*E);
    Asharp = Asharp/h; % to un-normalize Asharp
    A = A*h; % to un-normalize A

    % Calculate commanded gimbal angle rates
    deltaDotc = Asharp*u;
    mu = max(abs(deltaDotc));
    if mu >= deltaMax
        deltaDotc = deltaDotc*deltaMax/mu;
    end

    % Calculate derivatives
    wDot = -Jinv*(cross(w, J*w+hv) + A*deltaDot);
    qvDot = 0.5*(-cross(w, qv) + q(4)*w);
    q4Dot = -0.5*dot(w, qv);
    qDot = [qvDot; q4Dot];

    % Gimbal rate dynamics
    % Don't include gimbal dynamics:
    deltaDotArray(:,it+1) = deltaDotc;
    % Include gimbal dynamics:
%     deltaDotArray(:,it+1) = deltaDot + dt*deltaDot1D;
%     deltaDot2D = (50^2)*(deltaDotc - deltaDot) - 2*0.7*50*deltaDot1D;
%     deltaDot1D = deltaDot1D + dt*deltaDot2D;

    % Update quantities
    qArray(:,it+1) = q + dt*qDot;
    wArray(:,it+1) = w + dt*wDot;
    deltaArray(:,it+1) = delta + dt*deltaDot;

    % Calculate integral of error (Trapezoid Rule)
    if it == 1
        intEalt = err*dt/2;
    elseif max(abs(err)) < 0.01 && counter == 0
        % one-time reset of the integral when err gets small
        intEalt = err*dt/2; % toggle for trapezoid rule
%         intE = [0; 0; 0]; % toggle for rectangle rule
```

```
        counter = 1;
    else
        intEalt = intEalt + err*dt;
    end
    intE = intEalt - err*dt/2; % toggle for trapezoid rule
%     intE = intE + err*dt; % toggle for rectangle rule
end

% Save quantities
Wie.dt = dt;
Wie.tArray = tArray;
Wie.qArray = qArray;
Wie.wArray = wArray;
Wie.deltaArray = deltaArray;
Wie.deltaDotArray = deltaDotArray;

% Convert gimbal angles into degrees
deltaArray = deltaArray*180/pi;

% Convert angular velocity components into deg/s
wArray = wArray*180/pi;

% Plot quaternion components
plot(tArray, qArray(1,:))
title('q1')
figure
plot(tArray, qArray(2,:))
title('q2')
figure
plot(tArray, qArray(3,:))
title('q3')
figure
plot(tArray, qArray(4,:))
title('q4')

% Plot angular velocity components
% figure
% plot(tArray, wArray(1,:))
% title('w1'); ylabel('deg/s'); xlabel('seconds')
% figure
% plot(tArray, wArray(2,:))
% title('w2'); ylabel('deg/s'); xlabel('seconds')
% figure
% plot(tArray, wArray(3,:))
% title('w3'); ylabel('deg/s'); xlabel('seconds')

% Plot gimbal angles
% figure
% plot(tArray, deltaArray(1,:))
% title('delta1'); ylabel('degrees'); xlabel('seconds')
% figure
% plot(tArray, deltaArray(2,:))
% title('delta2'); ylabel('degrees'); xlabel('seconds')
% figure
% plot(tArray, deltaArray(3,:))
% title('delta3'); ylabel('degrees'); xlabel('seconds')
% figure
% plot(tArray, deltaArray(4,:))
% title('delta4'); ylabel('degrees'); xlabel('seconds')

% Plot gimbal angle rates
% figure
% plot(tArray, deltaDotArray(1,:))
% title('d/dt delta1'); ylabel('rad/s'); xlabel('seconds')
% figure
% plot(tArray, deltaDotArray(2,:))
```

```
% title('d/dt delta2'); ylabel('rad/s'); xlabel('seconds')
% figure
% plot(tArray, deltaDotArray(3,:))
% title('d/dt delta3'); ylabel('rad/s'); xlabel('seconds')
% figure
% plot(tArray, deltaDotArray(4,:))
% title('d/dt delta4'); ylabel('rad/s'); xlabel('seconds')



function output = Limiter2(err, J, slewMax, k, c, tauMax)
    output = [0; 0; 0];
    for it = 1:3
        output(it) = 0.5*(c/k)*min(sqrt(4*0.4*(tauMax/J(it,it))*abs(err(it))),slewMax);
    end
end


function out = sat(in, LB, UB)
% saturation function
if in > UB
    out = UB;
elseif in < LB
    out = LB;
else
    out = in;
end
end
```

Appendix D

Code for CMG Angular Momentum Vector

4-CMG system:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function hTotal = angVector4(h, beta, delta)
d1 = delta(1); d2 = delta(2);
d3 = delta(3); d4 = delta(4);
cb = cosd(beta); sb = sind(beta);
c1 = cosd(d1); s1 = sind(d1);
c2 = cosd(d2); s2 = sind(d2);
c3 = cosd(d3); s3 = sind(d3);
c4 = cosd(d4); s4 = sind(d4);
h1 = [-cb*s1; c1; sb*s1];
h2 = [-c2; -cb*s2; sb*s2];
h3 = [cb*s3; -c3; sb*s3];
h4 = [c4; cb*s4; sb*s4];
hTotal = h*(h1 + h2 + h3 + h4);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

3/4 CMG system:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function hTotal = angVector3(h, beta, delta)
d1 = delta(1); d2 = delta(2); d3 = delta(3);
cb = cosd(beta); sb = sind(beta);
c1 = cosd(d1); s1 = sind(d1);
c2 = cosd(d2); s2 = sind(d2);
c3 = cosd(d3); s3 = sind(d3);
h1 = [-cb*s1; c1; sb*s1];
h2 = [-c2; -cb*s2; sb*s2];
h3 = [cb*s3; -c3; sb*s3];
hTotal = h*(h1 + h2 + h3);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2/4 adjacent CMG system:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function hTotal = angVector2adj(h, beta, delta)
d1 = delta(1); d2 = delta(2);
cb = cosd(beta); sb = sind(beta);
c1 = cosd(d1); s1 = sind(d1);
c2 = cosd(d2); s2 = sind(d2);
h1 = [-cb*s1; c1; sb*s1];
h2 = [-c2; -cb*s2; sb*s2];
hTotal = h*(h1 + h2);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2/4 opposite CMG system:

%%%

```
function hTotal = angVector2opp(h, beta, delta)
d1 = delta(1); d3 = delta(2);
cb = cosd(beta); sb = sind(beta);
c1 = cosd(d1); s1 = sind(d1);
c3 = cosd(d3); s3 = sind(d3);
h1 = [-cb*s1; c1; sb*s1];
h3 = [cb*s3; -c3; sb*s3];
hTotal = h*(h1 + h3);
end
```

%%%

Appendix E

Code for Computing Jacobian

4-CMG system:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function A = Jacobian4(h, beta, delta)
d1 = delta(1); d2 = delta(2);
d3 = delta(3); d4 = delta(4);
cb = cosd(beta); sb = sind(beta);
c1 = cosd(d1); s1 = sind(d1);
c2 = cosd(d2); s2 = sind(d2);
c3 = cosd(d3); s3 = sind(d3);
c4 = cosd(d4); s4 = sind(d4);
A1 = [-cb*c1, s2, cb*c3, -s4];
A2 = [-s1, -cb*c2, s3, cb*c4];
A3 = [sb*c1, sb*c2, sb*c3, sb*c4];
A = h*[A1; A2; A3];
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

3/4 CMG system:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function A = Jacobian3(h, beta, delta)
d1 = delta(1); d2 = delta(2); d3 = delta(3);
cb = cosd(beta); sb = sind(beta);
c1 = cosd(d1); s1 = sind(d1);
c2 = cosd(d2); s2 = sind(d2);
c3 = cosd(d3); s3 = sind(d3);
A1 = [-cb*c1, s2, cb*c3];
A2 = [-s1, -cb*c2, s3];
A3 = [sb*c1, sb*c2, sb*c3];
A = h*[A1; A2; A3];
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2/4 adjacent CMG system:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function A = Jacobian2adj(h, beta, delta)
d1 = delta(1); d2 = delta(2);
cb = cosd(beta); sb = sind(beta);
c1 = cosd(d1); s1 = sind(d1);
c2 = cosd(d2); s2 = sind(d2);
A1 = [-cb*c1, s2];
A2 = [-s1, -cb*c2];
A3 = [sb*c1, sb*c2];
A = h*[A1; A2; A3];
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2/4 opposite CMG system:

%%%

```
function A = Jacobian2opp(h, beta, delta)
d1 = delta(1); d3 = delta(2);
cb = cosd(beta); sb = sind(beta);
c1 = cosd(d1); s1 = sind(d1);
c3 = cosd(d3); s3 = sind(d3);
A1 = [-cb*c1, cb*c3];
A2 = [-s1, s3];
A3 = [sb*c1, sb*c3];
A = h*[A1; A2; A3];
end
```

%%%

Appendix F

Code for Computing Angular Momentum Envelope

4-CMG system:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Compute angular momentum envelope for 4-CMG pyramid

close all
clear all

beta = 53.13;
h = 1;
dtheta = 15;
theta = 0:dtheta:360;
theta = theta(1:end-1);
N = length(theta);
points = zeros(3,N^4);
index = 1;
for it1 = theta
    for it2 = theta
        for it3 = theta
            for it4 = theta
                delta = [it1; it2; it3; it4];
                points(:,index) = angVector4(h, beta, delta);
                index = index + 1;
            end
        end
    end
end

x = reshape(points(1,:),N^2,N^2)';
y = reshape(points(2,:),N^2,N^2)';
z = reshape(points(3,:),N^2,N^2)';

% plot3(points(1,:),points(2,:),points(3,:))
surf(x, y, z)
xlim([-4.5, 4.5])
ylim([-4.5, 4.5])
zlim([-4.5, 4.5])
hAxis = gca;
hAxis.XRuler.FirstCrossoverValue  = 0; % X crossover with Y axis
hAxis.YRuler.FirstCrossoverValue  = 0; % Y crossover with X axis
hAxis.ZRuler.FirstCrossoverValue  = 0; % Z crossover with X axis
hAxis.ZRuler.SecondCrossoverValue = 0; % Z crossover with Y axis
hAxis.XRuler.SecondCrossoverValue = 0; % X crossover with Z axis
hAxis.YRuler.SecondCrossoverValue = 0; % Y crossover with Z axis
% xlabel('x')
% ylabel('y')
% zlabel('z')
view(37.5, 30)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

3/4 CMG system:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Compute angular momentum envelope for 3/4 pyramid

close all
clear all

beta = 53.13;
h = 1;
dtheta = 10;
theta = 0:dtheta:360;
theta = theta(1:end-1);
N = length(theta);
points = zeros(3,N^3);
index = 1;
for it1 = theta
    for it2 = theta
        for it3 = theta
            delta = [it1; it2; it3];
            points(:,index) = angVector3(h, beta, delta);
            index = index + 1;
        end
    end
end

x = reshape(points(1,:),216,216)';
y = reshape(points(2,:),216,216)';
z = reshape(points(3,:),216,216)';

% plot3(points(1,:),points(2,:),points(3,:))
surf(x, y, z)
xlim([-4.5, 4.5])
ylim([-4.5, 4.5])
zlim([-4.5, 4.5])
hAxis = gca;
hAxis.XRuler.FirstCrossoverValue  = 0; % X crossover with Y axis
hAxis.YRuler.FirstCrossoverValue  = 0; % Y crossover with X axis
hAxis.ZRuler.FirstCrossoverValue  = 0; % Z crossover with X axis
hAxis.ZRuler.SecondCrossoverValue = 0; % Z crossover with Y axis
hAxis.XRuler.SecondCrossoverValue = 0; % X crossover with Z axis
hAxis.YRuler.SecondCrossoverValue = 0; % Y crossover with Z axis
% xlabel('x')
% ylabel('y')
% zlabel('z')
view(37.5, 30)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2/4 adjacent CMG system:

%%%

```
% Compute angular momentum envelope for 2/4 adjacent pyramid

close all
clear all

beta = 53.13;
h = 1;
dtheta = 10;
theta = 0:dtheta:360;
theta = theta(1:end-1);
N = length(theta);
points = zeros(3,N^2);
index = 1;
for it1 = theta
    for it2 = theta
        delta = [it1; it2];
        points(:,index) = angVector2adj(h, beta, delta);
        index = index + 1;
    end
end

x = reshape(points(1,:),N,N)';
y = reshape(points(2,:),N,N)';
z = reshape(points(3,:),N,N)';

% plot3(points(1,:),points(2,:),points(3,:))
surf(x, y, z)
xlim([-4.5, 4.5])
ylim([-4.5, 4.5])
zlim([-4.5, 4.5])
hAxis = gca;
hAxis.XRuler.FirstCrossoverValue  = 0; % X crossover with Y axis
hAxis.YRuler.FirstCrossoverValue  = 0; % Y crossover with X axis
hAxis.ZRuler.FirstCrossoverValue  = 0; % Z crossover with X axis
hAxis.ZRuler.SecondCrossoverValue = 0; % Z crossover with Y axis
hAxis.XRuler.SecondCrossoverValue = 0; % X crossover with Z axis
hAxis.YRuler.SecondCrossoverValue = 0; % Y crossover with Z axis
% xlabel('x')
% ylabel('y')
% zlabel('z')
% [caz,cel] = view
view(37.5, 30)
```

%%%

2/4 opposite CMG system:

%%%

```
% Compute angular momentum envelope for 2/4 opposite pyramid

close all
clear all

beta = 53.13;
h = 1;
dtheta = 10;
theta = 0:dtheta:360;
theta = theta(1:end-1);
N = length(theta);
points = zeros(3,N^2);
index = 1;
for it1 = theta
    for it2 = theta
        delta = [it1; it2];
        points(:,index) = angVector2opp(h, beta, delta);
        index = index + 1;
    end
end

x = reshape(points(1,:),N,N)';
y = reshape(points(2,:),N,N)';
z = reshape(points(3,:),N,N)';

% plot3(points(1,:),points(2,:),points(3,:))
surf(x, y, z)
xlim([-4.5, 4.5])
ylim([-4.5, 4.5])
zlim([-4.5, 4.5])
hAxis = gca;
hAxis.XRuler.FirstCrossoverValue  = 0; % X crossover with Y axis
hAxis.YRuler.FirstCrossoverValue  = 0; % Y crossover with X axis
hAxis.ZRuler.FirstCrossoverValue  = 0; % Z crossover with X axis
hAxis.ZRuler.SecondCrossoverValue = 0; % Z crossover with Y axis
hAxis.XRuler.SecondCrossoverValue = 0; % X crossover with Z axis
hAxis.YRuler.SecondCrossoverValue = 0; % Y crossover with Z axis
% xlabel('x')
% ylabel('y')
% zlabel('z')
view(37.5, 30)
```

%%%

Appendix G

Code for Computing Vadali Preferred Angles

```
function delta = preferredX3(h, beta)
dt = 1/1000;
T = [1; 0; 0];
d1 = -88; % nominal -90
d2 = 180; % nominal 180
d3 = 88;  % nominal 90
delta = [d1; d2; d3];
while norm(angVector3(h, beta, delta)) > 0.00001*h
    Ddelta = inv(Jacobian3(h, beta, delta))*T;
    Ddelta = Ddelta*180/pi;
    delta = delta - dt*Ddelta;
%     norm(angVector3(h, beta, [d1; d2; d3]))
end
end
% Result:
% delta = [56.4; 180; -56.4];
% This corresponds to delta3 from Table 7 of Lee et al. (2018)
```

Appendix H

Code for Momentum Reduction Example

```
% This reduces the CMG configuration angular momentum
% to be close to zero, for the 2-CMG system


% Setup parameters
beta = 53.13; % skew angle, in degrees
h = 1000; % each CMG momentum, in SI units (constant)
deltaMax = 2; % max gimbal rate, in rad/s
slewMax = 10; % max slew rate, in deg/s
slewMax = slewMax*pi/180; % convert to rad/s
J = diag([21400,20100,5000]); % spacecraft tensor of inertia, in SI units
Jinv = inv(J);
tauMax = 100; % max external torque in each axis, in N-m
totalTime = 50; % total time of simulation, in seconds
Tcom = 5; % expected commanded time for maneuver
dt = 1/10000; % interval of time, in seconds

% Control parameters
omegaN = 3; % in rad/s
zeta = 0.9;
T = 1e1; % time constant of integral control, in seconds (should be 10)
k = omegaN^2 + 2*zeta*omegaN/T;
c = 2*zeta*omegaN + 1/T;

% Initial/final conditions
% q: spacecraft quaternion, w: spacecraft angular velocity (body frame),
% delta: gimbal angles, deltaDot: gimbal angle rates
q0 = [sind(5); 0; 0; cosd(5)]; % start 10 degrees off desired orientation
qf = [0; 0; 0; 1];
delta0 = [90; 90]; % Case 1
% delta0 = [-90; -90]; % Case 2
delta0 = delta0*pi/180; % convert to rad
deltaf = [90; -90]; % desired final gimbal angles
deltaf = deltaf*pi/180; % convert to rad
w0 = [0; 0; 0];
deltaDot1D = [0; 0]; % first derivative of deltaDot

% Quaternion error matrix
r1 = [qf(4), qf(3), -qf(2), -qf(1)];
r2 = [-qf(3), qf(4), qf(1), -qf(2)];
r3 = [qf(2), -qf(1), qf(4), -qf(3)];
r4 = [qf(1), qf(2), qf(3), qf(4)];
QEmatrix = [r1; r2; r3; r4];

% Quantities over time
tArray = 0:dt:totalTime;
N = length(tArray); % number of columns for each quantity
qArray = zeros(4,N);
wArray = zeros(3,N);
deltaArray = zeros(2,N);
deltaDotArray = zeros(2,N);

% Initialize
qArray(:,1) = q0;
wArray(:,1) = w0;
deltaArray(:,1) = delta0;
intE = [0; 0; 0]; % integral of error quaternion vector
counter = 0; % for one-time reset of integral


% Calculate maneuver simulation
for it = 1:N-1

    t = tArray(it);
    q = qArray(:,it);
    w = wArray(:,it);
```

```
    delta = deltaArray(:,it);
    deltaDot = deltaDotArray(:,it);
    qv = q(1:3);
    A = Jacobian2adj(h, beta, delta*180/pi);
    hv = angVector2adj(h, beta, delta*180/pi);

    % Calculate error quaternion vector
    qErr = QEmatrix*q;
    err = qErr(1:3);

    % Calculate saturated torque (tau)
    limitValues = Limiter2(err, J, slewMax, k, c, tauMax);
    errTerm = err + intE/T;
    satErr = [0; 0; 0];
    for it2 = 1:3
        satErr(it2) = sat(errTerm(it2), -limitValues(it2), limitValues(it2));
    end
    tau = -J*(2*k*satErr + c*w) + cross(w, hv) + A*deltaDot;

    % Compute commanded torque (tauc, external)
    tauc = tau;
    for it2 = 1:3
        tauc(it2) = sat(tauc(it2), -tauMax, tauMax);
    end

    % Calculate commanded gimbal angle rates
    deltaDotc = -(1/Tcom)*(delta - deltaf);
    mu = max(abs(A*deltaDotc));
    U = 0.7*tauMax;
    if mu >= U
        deltaDotc = deltaDotc*U/mu;
    end

    % Calculate derivatives
    wDot = -Jinv*(cross(w, J*w+hv) + A*deltaDot - tauc);
    qvDot = 0.5*(-cross(w, qv) + q(4)*w);
    q4Dot = -0.5*dot(w, qv);
    qDot = [qvDot; q4Dot];

    % Gimbal rate dynamics
    % Don't include gimbal dynamics:
    deltaDotArray(:,it+1) = deltaDotc;
    % Include gimbal dynamics:
%     deltaDotArray(:,it+1) = deltaDot + dt*deltaDot1D;
%     deltaDot2D = (50^2)*(deltaDotc - deltaDot) - 2*0.7*50*deltaDot1D;
%     deltaDot1D = deltaDot1D + dt*deltaDot2D;

    % Update quantities
    qArray(:,it+1) = q + dt*qDot;
    wArray(:,it+1) = w + dt*wDot;
    deltaArray(:,it+1) = delta + dt*deltaDot;

    % Calculate integral of error (Trapezoid Rule)
    if it == 1
        intEalt = err*dt/2;
    elseif max(abs(err)) < 0.01 && counter == 0
        % one-time reset of the integral when err gets small
        intEalt = err*dt/2; % toggle for trapezoid rule
%         intE = [0; 0; 0]; % toggle for rectangle rule
        counter = 1;
    else
        intEalt = intEalt + err*dt;
    end
    intE = intEalt - err*dt/2; % toggle for trapezoid rule
%     intE = intE + err*dt; % toggle for rectangle rule
end
```

```
% Save quantities
Wie.dt = dt;
Wie.tArray = tArray;
Wie.qArray = qArray;
Wie.wArray = wArray;
Wie.deltaArray = deltaArray;
Wie.deltaDotArray = deltaDotArray;

% Convert gimbal angles into degrees
deltaArray = deltaArray*180/pi;

% Convert angular velocity components into deg/s
wArray = wArray*180/pi;

% Plot quaternion components
plot(tArray, qArray(1,:))
title('q1')
figure
plot(tArray, qArray(2,:))
title('q2')
figure
plot(tArray, qArray(3,:))
title('q3')
figure
plot(tArray, qArray(4,:))
title('q4')

% Plot angular velocity components
% figure
% plot(tArray, wArray(1,:))
% title('w1'); ylabel('deg/s'); xlabel('seconds')
% figure
% plot(tArray, wArray(2,:))
% title('w2'); ylabel('deg/s'); xlabel('seconds')
% figure
% plot(tArray, wArray(3,:))
% title('w3'); ylabel('deg/s'); xlabel('seconds')

% Plot gimbal angles
figure
plot(tArray, deltaArray(1,:))
title('delta1'); ylabel('degrees'); xlabel('seconds')
figure
plot(tArray, deltaArray(2,:))
title('delta2'); ylabel('degrees'); xlabel('seconds')

% Plot gimbal angle rates
figure
plot(tArray, deltaDotArray(1,:))
title('d/dt delta1'); ylabel('rad/s'); xlabel('seconds')
figure
plot(tArray, deltaDotArray(2,:))
title('d/dt delta2'); ylabel('rad/s'); xlabel('seconds')



function output = Limiter2(err, J, slewMax, k, c, tauMax)
    output = [0; 0; 0];
    for it = 1:3
        output(it) = 0.5*(c/k)*min(sqrt(4*0.4*(tauMax/J(it,it))*abs(err(it))),slewMax);
    end
end
```

Appendix I

Code for Computing Gauss Nodes, Weights, and Differential Approximation Matrix

```
%---------------------------------------------------------------------------
% Gauss_nodes.m
% determines Gauss nodes
%---------------------------------------------------------------------------
% tau = Gauss_nodes(N)
%   N: number of nodes minus 1, should be an integer greater than 0
% tau: Gauss nodes
%---------------------------------------------------------------------------
% Examples:
% tau = Gauss_nodes(1)
% 0
% tau = Gauss_nodes(2)
% -0.57735 0.57735
% tau = Gauss_nodes(3)
% -0.77460 0 0.77460
%---------------------------------------------------------------------------
% Contributor: Victor Hakim
%---------------------------------------------------------------------------
function tau = Gauss_nodes(N)

if N == 9
    run('Gauss9.m')
    tau = tau9;
elseif N == 19
    run('Gauss19.m')
    tau = tau19;
elseif N == 49
    run('Gauss49.m')
    tau = tau49;
elseif N == 99
    run('Gauss99.m')
    tau = tau99;
else

    syms z;
    tau = sort(double(vpasolve(legendreP(N,z) == 0)));

end
end
```

```
%--------------------------------------------------------------------------
% Gauss_weights.m
% determines Gaussian quadrature weights using Gauss nodes
%--------------------------------------------------------------------------
% w = Gauss_weights(tau)
% tau: Gauss nodes
%   w: Gaussian quadrature weights
%--------------------------------------------------------------------------
% Author: Daniel R. Herber, Graduate Student, University of Illinois at
% Urbana-Champaign
% Date: 06/04/2015
%--------------------------------------------------------------------------
function w = Gauss_weights(tau)
% number of nodes
N = length(tau);

if N == 9
    run('Gauss9.m')
    w = w9;
elseif N == 19
    run('Gauss19.m')
    w = w19;
elseif N == 49
    run('Gauss49.m')
    w = w49;
elseif N == 99
    run('Gauss99.m')
    w = w99;
else

    % See Benson's MIT PhD thesis, page 30. Note the typo.
    [der, ~] = lepoly(N, tau);
    w = 2./((1 - tau.^2).*der.^2);

end
end
```

```
%-------------------------------------------------------------------------
% Gauss_Dmatrix.m
% determines approximate differentiation matrix for Gauss pseudospectral
% method with Gauss points
%-------------------------------------------------------------------------
% D = Gauss_Dmatrix(tau)
% tau: Gauss nodes, length N
%   D: differentiation matrix, dimensions Nx(N+1)
%-------------------------------------------------------------------------
% Author: Victor Hakim
%-------------------------------------------------------------------------
function D = Gauss_Dmatrix(tau)
% uses the function LagrangePoly(), function of symbolic z

N = length(tau);

if N == 9
    run('Gauss9.m')
    D = D9;
elseif N == 19
    run('Gauss19.m')
    D = D19;
elseif N == 49
    run('Gauss49.m')
    D = D49;
elseif N == 99
    run('Gauss99.m')
    D = D99;
else

    dLagr = diff(LagrangePoly(cat(1, -1, tau)));
    for it1 = 1:N
        for it2 = 0:N
            z = tau(it1);
            D(it1, it2+1) = subs(dLagr(it2+1));
        end
    end

end
end
```

Appendix J

Code for Generating Plots

```
%-------------------------------------------------------------------------
% Plots2_Gauss_CMG3.m
% Plotting function, Gauss pseudospectral method
% For 3/4 CMG pyramid configuration
%-------------------------------------------------------------------------
%
%-------------------------------------------------------------------------
% Underlying code: Daniel R. Herber, Graduate Student, University of
% Illinois at Urbana-Champaign
% https://github.com/danielrherber/optimal-control-direct-method-examples
%-------------------------------------------------------------------------
function Plots2_Gauss_CMG3(q,w,delta,u,Det,HSystem,p,Wie,method)
    close all
    %-------------------------------------------------------------------------
    fontlabel = 26; % x,y label font size
    fontlegend = 21; % x,y legend font size
    fonttick = 21; % x,y tick font size
    wcolor = [1 1 1]; % white color
    bcolor = [0 0 0]; % black color
    mycolor = lines(8);

    set(0,'DefaultTextInterpreter','latex'); % change the text interpreter
    set(0,'DefaultLegendInterpreter','latex'); % change the legend interpreter
    set(0,'DefaultAxesTickLabelInterpreter','latex'); % change the tick interpreter

    set(0, 'DefaultLineLineWidth', 1.2);

    T = linspace(p.t0,p.tf,10000);
    longt = [p.t0; p.t];
%     longert = [longt; p.tf];

    if strcmp(method,'Pseudospectral')
        q1l = LagrangeInter(longt',q(:,1)',T);
        q2l = LagrangeInter(longt',q(:,2)',T);
        q3l = LagrangeInter(longt',q(:,3)',T);
        q4l = LagrangeInter(longt',q(:,4)',T);
        w1l = LagrangeInter(longt',w(:,1)',T);
        w2l = LagrangeInter(longt',w(:,2)',T);
        w3l = LagrangeInter(longt',w(:,3)',T);
        d1l = LagrangeInter(longt',delta(:,1)',T);
        d2l = LagrangeInter(longt',delta(:,2)',T);
        d3l = LagrangeInter(longt',delta(:,3)',T);
        u1l = LagrangeInter(p.t',u(:,1)',T);
        u2l = LagrangeInter(p.t',u(:,2)',T);
        u3l = LagrangeInter(p.t',u(:,3)',T);
        Detl = LagrangeInter(longt',Det',T);
        HSysteml = LagrangeInter(longt',HSystem',T);
    end


    %-------------------------------------------------------------------------
    % Plot Q1
    hf = figure; % create a new figure and save handle
    hf.Color = wcolor; % change the figure background color

    plot(longt,q(:,1),'.','markersize',14,'color',mycolor(1,:)); hold on

    plot(Wie.tArray,Wie.qArray(1,:),'color',mycolor(7,:)); hold on

    if strcmp(method,'Pseudospectral')
        plot(T,q1l,'-','color',mycolor(1,:)); hold on
    end

    mytitle = ['']; % title with latex
    myxlabel = '$t$ (sec.)'; % x label with latex
    myylabel = '$q_{_1}$'; % y label with latex
```

```
    mylegend = {'Gauss PS Method','Generalized SR'}; % legend with latex
    xlabel(myxlabel) % create x label
    ylabel(myylabel) % create y label
    ha = gca; % get current axis handle
    try
        ha.XAxis.Color = bcolor; % change the x axis color to black (not a dark grey)
        ha.XAxis.FontSize = fonttick; % change x tick font size
        ha.YAxis.FontSize = fonttick; % change y tick font size
        ha.XAxis.Label.FontSize = fontlabel; % change x label font size
        ha.YAxis.Label.FontSize = fontlabel; % change y label font size
        ht = title(mytitle);
        ht.FontSize = fontlabel;
        hl = legend(mylegend,'location','Best'); % create legend
        hl.FontSize = fontlegend; % change legend font size
        hl.EdgeColor = bcolor; % change the legend border to black (not a dark grey)
    catch
        disp('plot formatting failed (try using a version that supports HG2)')
    end

    %-----------------------------------------------------------------------
    % Plot Q2
    hf = figure; % create a new figure and save handle
    hf.Color = wcolor; % change the figure background color

    plot(longt,q(:,2),'.','markersize',14,'color',mycolor(1,:)); hold on

    plot(Wie.tArray,Wie.qArray(2,:),'color',mycolor(7,:)); hold on

    if strcmp(method,'Pseudospectral')
        plot(T,q2l,'-','color',mycolor(1,:)); hold on
    end

%     mytitle = ['$q_2$']; % title with latex
    myxlabel = '$t$ (sec.)'; % x label with latex
    myylabel = '$q_{_2}$'; % y label with latex
    mylegend = {'Gauss PS Method','Generalized SR'}; % legend with latex
    xlabel(myxlabel) % create x label
    ylabel(myylabel) % create y label
    ha = gca; % get current axis handle
    try
        ha.XAxis.Color = bcolor; % change the x axis color to black (not a dark grey)
        ha.XAxis.FontSize = fonttick; % change x tick font size
        ha.YAxis.FontSize = fonttick; % change y tick font size
        ha.XAxis.Label.FontSize = fontlabel; % change x label font size
        ha.YAxis.Label.FontSize = fontlabel; % change y label font size
        ht = title(mytitle);
        ht.FontSize = fontlabel;
        hl = legend(mylegend,'location','Best'); % create legend
        hl.FontSize = fontlegend; % change legend font size
        hl.EdgeColor = bcolor; % change the legend border to black (not a dark grey)
    catch
        disp('plot formatting failed (try using a version that supports HG2)')
    end

    %-----------------------------------------------------------------------
    % Plot Q3
    hf = figure; % create a new figure and save handle
    hf.Color = wcolor; % change the figure background color

    plot(longt,q(:,3),'.','markersize',14,'color',mycolor(1,:)); hold on

    plot(Wie.tArray,Wie.qArray(3,:),'color',mycolor(7,:)); hold on

    if strcmp(method,'Pseudospectral')
        plot(T,q3l,'-','color',mycolor(1,:)); hold on
    end
```

```
%     mytitle = ['$q_3$']; % title with latex
    myxlabel = '$t$ (sec.)'; % x label with latex
    myylabel = '$q_{_3}$'; % y label with latex
    mylegend = {'Gauss PS Method','Generalized SR'}; % legend with latex
    xlabel(myxlabel) % create x label
    ylabel(myylabel) % create y label
    ha = gca; % get current axis handle
    try
        ha.XAxis.Color = bcolor; % change the x axis color to black (not a dark grey)
        ha.XAxis.FontSize = fonttick; % change x tick font size
        ha.YAxis.FontSize = fonttick; % change y tick font size
        ha.XAxis.Label.FontSize = fontlabel; % change x label font size
        ha.YAxis.Label.FontSize = fontlabel; % change y label font size
        ht = title(mytitle);
        ht.FontSize = fontlabel;
        hl = legend(mylegend,'location','Best'); % create legend
        hl.FontSize = fontlegend; % change legend font size
        hl.EdgeColor = bcolor; % change the legend border to black (not a dark grey)
    catch
        disp('plot formatting failed (try using a version that supports HG2)')
    end

    %-----------------------------------------------------------------------
    % Plot Q4
    hf = figure; % create a new figure and save handle
    hf.Color = wcolor; % change the figure background color

    plot(longt,q(:,4),'.','markersize',14,'color',mycolor(1,:)); hold on

    plot(Wie.tArray,Wie.qArray(4,:),'color',mycolor(7,:)); hold on

    if strcmp(method,'Pseudospectral')
        plot(T,q4l,'-','color',mycolor(1,:)); hold on
    end

%     mytitle = ['$q_4$']; % title with latex
    myxlabel = '$t$ (sec.)'; % x label with latex
    myylabel = '$q_{_4}$'; % y label with latex
    mylegend = {'Gauss PS Method','Generalized SR'}; % legend with latex
    xlabel(myxlabel) % create x label
    ylabel(myylabel) % create y label
    ha = gca; % get current axis handle
    try
        ha.XAxis.Color = bcolor; % change the x axis color to black (not a dark grey)
        ha.XAxis.FontSize = fonttick; % change x tick font size
        ha.YAxis.FontSize = fonttick; % change y tick font size
        ha.XAxis.Label.FontSize = fontlabel; % change x label font size
        ha.YAxis.Label.FontSize = fontlabel; % change y label font size
        ht = title(mytitle);
        ht.FontSize = fontlabel;
        hl = legend(mylegend,'location','Best'); % create legend
        hl.FontSize = fontlegend; % change legend font size
        hl.EdgeColor = bcolor; % change the legend border to black (not a dark grey)
    catch
        disp('plot formatting failed (try using a version that supports HG2)')
    end


    %-----------------------------------------------------------------------
    % Plot w1
    hf = figure; % create a new figure and save handle
    hf.Color = wcolor; % change the figure background color

    plot(longt,w(:,1),'.','markersize',14,'color',mycolor(6,:)); hold on
```

```
    plot(Wie.tArray,Wie.wArray(1,:),'color',mycolor(2,:)); hold on

    if strcmp(method,'Pseudospectral')
        plot(T,w11,'-','color',mycolor(6,:)); hold on
    end

%     mytitle = ['$\omega_1$']; % title with latex
    myxlabel = '$t$ (sec.)'; % x label with latex
    myylabel = '$\omega_1$ - rad/sec'; % y label with latex
    mylegend = {'Gauss PS Method','Generalized SR'}; % legend with latex
    xlabel(myxlabel) % create x label
    ylabel(myylabel) % create y label
    ha = gca; % get current axis handle
    try
        ha.XAxis.Color = bcolor; % change the x axis color to black (not a dark grey)
        ha.XAxis.FontSize = fonttick; % change x tick font size
        ha.YAxis.FontSize = fonttick; % change y tick font size
        ha.XAxis.Label.FontSize = fontlabel; % change x label font size
        ha.YAxis.Label.FontSize = fontlabel; % change y label font size
        ht = title(mytitle);
        ht.FontSize = fontlabel;
        hl = legend(mylegend,'location','Best'); % create legend
        hl.FontSize = fontlegend; % change legend font size
        hl.EdgeColor = bcolor; % change the legend border to black (not a dark grey)
    catch
        disp('plot formatting failed (try using a version that supports HG2)')
    end

    %-----------------------------------------------------------------------
    % Plot w2
    hf = figure; % create a new figure and save handle
    hf.Color = wcolor; % change the figure background color

    plot(longt,w(:,2),'.','markersize',14,'color',mycolor(6,:)); hold on

    plot(Wie.tArray,Wie.wArray(2,:),'color',mycolor(2,:)); hold on

    if strcmp(method,'Pseudospectral')
        plot(T,w21,'-','color',mycolor(6,:)); hold on
    end

%     mytitle = ['$\omega_2$']; % title with latex
    myxlabel = '$t$ (sec.)'; % x label with latex
    myylabel = '$\omega_2$ - rad/sec'; % y label with latex
    mylegend = {'Gauss PS Method','Generalized SR'}; % legend with latex
    xlabel(myxlabel) % create x label
    ylabel(myylabel) % create y label
    ha = gca; % get current axis handle
    try
        ha.XAxis.Color = bcolor; % change the x axis color to black (not a dark grey)
        ha.XAxis.FontSize = fonttick; % change x tick font size
        ha.YAxis.FontSize = fonttick; % change y tick font size
        ha.XAxis.Label.FontSize = fontlabel; % change x label font size
        ha.YAxis.Label.FontSize = fontlabel; % change y label font size
        ht = title(mytitle);
        ht.FontSize = fontlabel;
        hl = legend(mylegend,'location','Best'); % create legend
        hl.FontSize = fontlegend; % change legend font size
        hl.EdgeColor = bcolor; % change the legend border to black (not a dark grey)
    catch
        disp('plot formatting failed (try using a version that supports HG2)')
    end

    %-----------------------------------------------------------------------
    % Plot w3
    hf = figure; % create a new figure and save handle
```

```
    hf.Color = wcolor; % change the figure background color

    plot(longt,w(:,3),'.','markersize',14,'color',mycolor(6,:)); hold on

    plot(Wie.tArray,Wie.wArray(3,:),'color',mycolor(2,:)); hold on

    if strcmp(method,'Pseudospectral')
        plot(T,w3l,'-','color',mycolor(6,:)); hold on
    end

%     mytitle = ['$\omega_3$']; % title with latex
    myxlabel = '$t$ (sec.)'; % x label with latex
    myylabel = '$\omega_3$ - rad/sec'; % y label with latex
    mylegend = {'Gauss PS Method','Generalized SR'}; % legend with latex
    xlabel(myxlabel) % create x label
    ylabel(myylabel) % create y label
    ha = gca; % get current axis handle
    try
        ha.XAxis.Color = bcolor; % change the x axis color to black (not a dark grey)
        ha.XAxis.FontSize = fonttick; % change x tick font size
        ha.YAxis.FontSize = fonttick; % change y tick font size
        ha.XAxis.Label.FontSize = fontlabel; % change x label font size
        ha.YAxis.Label.FontSize = fontlabel; % change y label font size
        ht = title(mytitle);
        ht.FontSize = fontlabel;
        hl = legend(mylegend,'location','Best'); % create legend
        hl.FontSize = fontlegend; % change legend font size
        hl.EdgeColor = bcolor; % change the legend border to black (not a dark grey)
    catch
        disp('plot formatting failed (try using a version that supports HG2)')
    end


    %-----------------------------------------------------------------------
    % Plot gimbal angles
    hf = figure; % create a new figure and save handle
    hf.Color = wcolor; % change the figure background color

    % Convert to degrees
    delta = delta*180/pi;
    d1l = d1l*180/pi;
    d2l = d2l*180/pi;
    d3l = d3l*180/pi;

    if strcmp(method,'Pseudospectral')
        plot(T,d1l,'-','color',mycolor(1,:)); hold on
        plot(T,d2l,':','color',mycolor(4,:),'linewidth',2); hold on
        plot(T,d3l,'--','color',mycolor(2,:)); hold on
    end

    plot(longt,delta(:,1),'.','markersize',14,'color',mycolor(1,:)); hold on
    plot(longt,delta(:,2),'.','markersize',14,'color',mycolor(4,:)); hold on
    plot(longt,delta(:,3),'.','markersize',14,'color',mycolor(2,:)); hold on

%     mytitle = ['Gimbal Angles']; % title with latex
    myxlabel = '$t$ (sec.)'; % x label with latex
    myylabel = 'Gimbal angles - degrees'; % y label with latex
    mylegend = {'$\delta_1$','$\delta_2$','$\delta_3$'}; % legend with latex
    xlabel(myxlabel) % create x label
    ylabel(myylabel) % create y label
    ha = gca; % get current axis handle
    try
        ha.XAxis.Color = bcolor; % change the x axis color to black (not a dark grey)
        ha.XAxis.FontSize = fonttick; % change x tick font size
        ha.YAxis.FontSize = fonttick; % change y tick font size
        ha.XAxis.Label.FontSize = fontlabel; % change x label font size
```

```
        ha.YAxis.Label.FontSize = fontlabel; % change y label font size
        ht = title(mytitle);
        ht.FontSize = fontlabel;
        hl = legend(mylegend,'location','Northwest'); % create legend
        hl.FontSize = fontlegend; % change legend font size
        hl.EdgeColor = bcolor; % change the legend border to black (not a dark grey)
    catch
        disp('plot formatting failed (try using a version that supports HG2)')
    end


    %----------------------------------------------------------------------
    % Plot Jacobian determinant
    hf = figure; % create a new figure and save handle
    hf.Color = wcolor; % change the figure background color

    plot(longt,Det/(p.h^3),'.','markersize',14,'color',mycolor(7,:)); hold on

    if strcmp(method,'Pseudospectral')
        plot(T,Detl/(p.h^3),'-','color',mycolor(7,:)); hold on
    end

%     mytitle = ['Determinant of Jacobian']; % title with latex
    myxlabel = '$t$ (sec.)'; % x label with latex
    myylabel = '$\det(\tilde{A})$'; % y label with latex
%     myylabel = '';
    xlabel(myxlabel) % create x label
    ylabel(myylabel) % create y label
    ylim([-1,1])
    ha = gca; % get current axis handle
    try
        ha.XAxis.Color = bcolor; % change the x axis color to black (not a dark grey)
        ha.XAxis.FontSize = fonttick; % change x tick font size
        ha.YAxis.FontSize = fonttick; % change y tick font size
        ha.XAxis.Label.FontSize = fontlabel; % change x label font size
        ha.YAxis.Label.FontSize = fontlabel; % change y label font size
        ht = title(mytitle);
        ht.FontSize = fontlabel;
    catch
        disp('plot formatting failed (try using a version that supports HG2)')
    end


    %----------------------------------------------------------------------
    % Plot system angular momentum magnitude
    hf = figure; % create a new figure and save handle
    hf.Color = wcolor; % change the figure background color

    plot(longt,HSystem,'.','markersize',14,'color',mycolor(7,:)); hold on

    if strcmp(method,'Pseudospectral')
        plot(T,HSystem1,'-','color',mycolor(7,:)); hold on
    end

    maxH = max(HSystem1);

    mytitle = ['System Angular Momentum Magnitude']; % title with latex
    myxlabel = '$t$ (sec.)'; % x label with latex
    myylabel = 'N$\cdot$m$\cdot$s'; % y label with latex
%     myylabel = '';
    xlabel(myxlabel) % create x label
    ylabel(myylabel) % create y label
    ylim([-0.1*maxH, 1.2*maxH])
    ha = gca; % get current axis handle
    try
        ha.XAxis.Color = bcolor; % change the x axis color to black (not a dark grey)
```

```
        ha.XAxis.FontSize = fonttick; % change x tick font size
        ha.YAxis.FontSize = fonttick; % change y tick font size
        ha.XAxis.Label.FontSize = fontlabel; % change x label font size
        ha.YAxis.Label.FontSize = fontlabel; % change y label font size
        ht = title(mytitle);
        ht.FontSize = fontlabel;
    catch
        disp('plot formatting failed (try using a version that supports HG2)')
    end




    %--------------------------------------------------------------------
    % Plot quaternion magnitude
    hf = figure; % create a new figure and save handle
    hf.Color = wcolor; % change the figure background color

    qMag = sqrt(q1l.^2 + q2l.^2 + q3l.^2 + q4l.^2);
    plot(T,qMag,'-','color',mycolor(7,:)); hold on

    mytitle = ['Quaternion Magnitude']; % title with latex
    myxlabel = '$t$ (sec.)'; % x label with latex
    myylabel = 'Magnitude'; % y label with latex
    xlabel(myxlabel) % create x label
    ylabel(myylabel) % create y label
    ylim([0 1.3])
    ha = gca; % get current axis handle
    try
        ha.XAxis.Color = bcolor; % change the x axis color to black (not a dark grey)
        ha.XAxis.FontSize = fonttick; % change x tick font size
        ha.YAxis.FontSize = fonttick; % change y tick font size
        ha.XAxis.Label.FontSize = fontlabel; % change x label font size
        ha.YAxis.Label.FontSize = fontlabel; % change y label font size
        ht = title(mytitle);
        ht.FontSize = fontlabel;
    catch
        disp('plot formatting failed (try using a version that supports HG2)')
    end


end
```

Vita

In 2017, Victor Hakim obtained his Bachelor's Degree from the University of California, Berkeley in Applied Mathematics with a focus on Quantum Mechanics. He transitioned his studies into Mechanical Engineering with a Dynamics and Controls focus in his Master's graduate studies at Santa Clara University. He enjoys doing research in spacecraft attitude control, specifically CMG steering laws. Beginning summer 2021, he will start as a Guidance, Navigation, Control (GNC) Engineer at SpaceX in Hawthorne, California.